

实用技术：开发Linux应用

——用GTK+和GDK开发Linux图形用户界面应用

〔美〕Eric Harlow 著

童寿彬 童 悦 冯 艺 等译

Developing Linux Applications with GTK+ and GDK

New Riders

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL: http://www.phei.com.cn

实用技术:开发 Linux 应用

——用 GTK+和 GDK 开发 Linux 图形用户界面应用

Developing Linux Applications with GTK+ and GDK

[美] Eric Harlow 著

童寿彬 童 悦 冯 艺 等译

TS79/14

電子工業出版社

Publishing House of Electronics Industry

北京·BEIJING

Authorized translation from the English language edition published by NEW RIDERS, an imprint of Macmillan Computer Publishing U. S. A.
Copyright © 1999
All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.
SIMPLIFIED CHINESE language edition published by Publishing House of Electronics Industry, China.
Copyright © 2000

本书中文简体专有翻译出版权由美国 Macmillan Computer Publishing 下属的 NEW RIDERS 授予电子工业出版社，该专有出版权受法律保护。

图书在版编目(CIP)数据

实用技术：开发 Linux 应用——用 GTK + 和 GDK 开发 Linux 图形用户界面应用/
(美)哈洛(Harlow, E.)著；　童寿彬等译. --北京：电子工业出版社，2000.1
书名原文：Developing Linux Applications with GTK + and GDK
ISBN 7-5053-5680-1

Ⅰ. 实… Ⅱ. ①哈… ②童… Ⅲ. 操作系统(软件)、Linux 软件开发-应用软件，GTK、GDK Ⅳ. TP317

中国版本图书馆 CIP 数据核字(1999)第 39165 号

书　　名：**实用技术：开发 Linux 应用——用 GTK + 和 GDK 开发 Linux 图形用户界面应用**
原 书 名：Developing Linux Applications with GTK + and GDK
著　　者：[美] Eric Harlow
译　　者：童寿彬　童　悦　冯　艺　等
责任编辑：周宏敏
排版制作：电子工业出版社计算机排版室监制
印 刷 者：北京天竺颖华印刷厂
装 订 者：三河市金马印装有限公司
出版发行：电子工业出版社　URL：http://www.phei.com.cn
　　北京市海淀区万寿路 173 信箱　邮编 100036
经　　销：各地新华书店
开　　本：787×1092　1/16　印张：31　字数：794 千字
版　　次：2000 年 1 月第 1 版　2000 年 1 月第 1 次印刷
书　　号：ISBN 7-5053-5680-1 / TP·2921
印　　数：4000 册　　定价：58.00 元
版权贸易合同登记号　图字：01-1999-1176

凡购买电子工业出版社的图书，如有缺页、倒页、脱页、所附磁盘或光盘有问题者，请向购买书店调换
若书店售缺，请与本社发行部联系调换。电话　68279077

译者的话

许多人熟悉 Windows 上的图像编辑软件 Photoshop。对它的强大功能和灵活性留下深刻的印象。其实，在 UNIX 类操作系统上也有类似的软件，它的名称叫 GIMP。GIMP 是通用影像处理程序(General Image Manipulation Program)或者 GNU 影像处理程序(GNU Image Manipulation Program)的缩写。它的功能和风格类似 Photoshop。它是开放源代码软件，用户可以从 Internet 上自由下载 GIMP 和它的大量插入模块，用户也可以根据需要开发自己的插入模块。开发 GIMP 使用了 GTK(GIMP toolkit，GIMP 工具集)。本书将全面介绍 GTK。

虽然 GTK 最早用来开发 GIMP，目前，GTK 已用在许多软件开发项目中。它和 UNIX 类操作系统和 X Window 一起使用，用来开发图形用户界面(GUI)应用。它也属于开放源代码软件，允许用户按 LGPL(库通用公共许可证)条款使用它。因此，可以用它开发自由软件或者商业软件而不必为许可证或者专利付任何费用。

GTK 为许多高级语言提供接口，其中包括 Ada95、C++、Pascal、Perl 等。在支持这些语言的 UNIX 类系统上都能使用它。

随着在过去几年中 Linux 的爆炸性增长，对开发高质量的 GUI 应用的需求达到了前所未有的程度。在 Linux 系统上，曾经有许多计算机高手在 Internet 上为开发 GUI 应用提供过不同的工具集，但是由于缺乏支持者而停滞不前。GTK 的出现，使情况发生了根本的变化。它得到了广泛的支持，大量用于开发 Linux 的 GUI 应用。

GTK 是新的、具有独创性的方法。利用这一应用编程界面将大大提高编程效率。使用 GTK 将使 Linux 的应用开发迈向新的水平。

介绍 GTK 的书不多。各方面对 New Riders 出版的《开发 Linux 应用》的评价很高。

本书是为开发 GUI 应用的 Linux 开发人员写的。假定读者熟悉 C 语言，并对 Linux 有一定了解。本书同样适用于从其他操作系统(如 MS Windows)转到 Linux 系统的开发 GUI 应用的人员。

参加本书翻译工作的人员包括：童寿彬、童悦、冯艺、叶慧文、刘承伟、薛怀成、郑立等。译者水平有限，不当之处请读者指正。

作者简介

作者 Eric Harlow 从 Delaware 大学获得计算机科学学士学位。他在 Vic-20 计算机上开始编程，在许多计算机上工作过，其中包括 Amiga、Atari ST 和个人计算机。他的操作系统经验包括 UNIX、Windows、OS/2，当然还包括 Linux。他开发过许多应用，其中包括计算机游戏，商业应用和 Web 站点。

Eric 当前是马里兰州巴尔的摩 RDA 顾问公司的顾问。他开发客户/服务器应用和基于 Web 的应用。他在工作之余忙于钻研 Linux 和 java。他对 Linux 的贡献有：用基于 Linux 的 ISP 作为例子说明这一操作系统的强大功能。

他全家现在住在巴尔的摩。

技术审核者的情况

在撰写《开发 Linux 应用》一书的整个过程中，审核者奉献了他亲身感受的大量实际经验。在写书的过程中，他对书的技术内容、素材的组织和可用性进行了全面的审核。他的反馈意见对保证《开发 Linux 应用》一书满足高质量技术知识的需求起了重要作用。

Thomas Mailund Jensen 是丹麦 Aarhus 大学计算机科学专业的学生。作为 CPN(Coloured Petri Nets)集团的成员，他正在完成他的哲学博士论文。他也是 CPN 的兼职程序员，从事 UNIX(包括 Linux)系统应用的开发。他的兴趣包括模型或系统的检验和证明，包括对已经发布的系统和算法的验证。为什么他钻研 GTK+？“纯粹出于对 GTK+的爱好。”

鸣谢

写书是一项烦琐而耗费时间的任务，没有我妻子 Emily 的理解就不可能写成这本书。她给我写书所需要的时间，包括许多周末和深夜在地下计算机前度过的时光。她理解我是计算机迷和 Linux 拥护者，尽管争取世界范围的软件控制权的战斗仍然举步艰难。

在 Macmillan 公司方面，感谢 Laurie Petrycki 提供写书的机会。此外，还得感谢开发编辑 Jim Chalex 在写书过程中给予的指导。看到成书的完整过程使我大开眼界。还得感谢技术审核 Thomas Mailund Jensen 在审核过程中提出的宝贵意见。

感谢 RDA 顾问公司的大老板 Don Awalt。特别感谢项目经理 Brian Berns，在一些重要的周末他照料了项目，使我有时间在家写书赶进度。

必须感谢 GTK+和 GIMP 的开发人员以及管理 GTK+邮递表的开发人员，他们回答了我提出的所有问题，使写成本书成为可能。

特别感谢 Linus Torvalds 和其他为 Linux 发展作出贡献的人们，他们发动了争取世界范围的软件控制权的斗争。感谢 slashdot.org，在开始这一项目时允许我使用它的站点作为论坛，并使 Linux 团体的每个人得到有关 Linux 和开放源代码软件方面进展情况的通知。感谢读过本书早期 beta 版的读者返回的改正意见。

感谢购买本书的读者，感谢他们对本书不完善的地方的宽容。写书要做到没有错误是很难的，让我们共同从中学习。

最后，还得感谢我的双亲。

请将你的想法告诉我们

作为本书的读者，你是我们最重要的批评家和评论家。我们尊重你的意见。我们需要知道哪些是做对了的，哪些需要改进，你希望我们在哪些领域出版书籍，以及其他方面的意见。

作为 New Riders 公司负责出版 Linux 专题的执行编辑，欢迎你提出意见。你可以用传真、电子邮件或直接写信的方式，让我知道你喜欢本书的哪些部分或不喜欢哪些部分。还有我们可以在哪些方面作改进。

请注意：我不能帮助解决与本书的专题有关的技术问题。此外由于收到邮件的数量大，我可能并不能对来信一一作出答复。当你写信时，请一定写上书名和作者，以及你的姓名和电话或传真号码。我将仔细阅读你的意见，并将这些意见告诉本书的作者和编辑。

Fax：317-581-4663

Email：newriders@mcp.com

Mail：Laurie Petrycki

New Riders Publishing

201 West 103rd Street

Indianapolis，IN 46290 USA

第一部分　用 GTK + 编程

第 1 章　介绍 GTK +
第 2 章　GLIB
第 3 章　开发 GUI 应用
第 4 章　基本构件
第 5 章　菜单、工具条和工具提示
第 6 章　更多的构件

第 1 章　介绍 GTK +

GIMP 是 UNIX 类操作系统上的影像处理程序。GIMP 是通用影像处理程序(General Image Manipulation Program)或者 GNU 影像处理程序(GNU Image Manipulation Program)的缩写。它的功能和风格类似 Windows 的 Photoshop。它是开放源代码软件,可以从 Web 站点 www.gimp.org 上得到其源代码。

GTK + 表示 GIMP 工具集(GIMP Toolkit,GTK)。GTK + 是工具库,最早用于 GIMP 的开发。这是为什么将 GTK + 称为 GIMP 工具集的原因。GIMP 也成为由专业人员使用 GTK + 开发的图形用户界面应用的范例。

虽然 GTK + 最早用于 GIMP 的开发,但它现在广泛用于开发 Linux 的图形用户界面(GUI)应用。GTK + 是开放源代码的工具库。用户可以按 LGPL(库通用公共许可证)的条款使用它。在其他支持 GTK + 库的平台上也可以用它来开发 GUI 应用。

GTK + 库是用 C 语言写成的面向对象的库,它支持用多种语言编写的应用软件。支持的语言包括:Ada95、C ++ 、Eiffel、Objective C、Pascal、Perl、Python 等。可以从 GTK + 的 Web 站点(www.gtk.org)查到支持的语言的最新清单。

1.1　对读者的要求

假定读者熟悉 C 语言,并对 Linux 有一定了解。读者应该掌握指针和内存分配的使用,还应该熟悉开发 GUI 应用软件和基于事件的消息传递。简单讲,本书是为开发 GUI 应用的 Linux 开发人员编写的。

本书同样适用于从其他操作系统(如 MS Windows)转到 Linux 系统的开发 GUI 应用软件的人员。

1.2　本书的内容

本书讨论 GTK + 库,包括 GLIB 和用 C 语言写的 GDK。这并不是说在本书中包含 GTK + 的所有内容。它的内容太多,不可能在一本书中容纳所有的主题。本书的目标是让开发人员了解 GTK +:它是在 Linux 系统和其他操作系统上用来开发 GUI 应用软件的工具集。

GTK + 库属于自由软件,或者说它是开放源代码的软件。所以可下载整个库的源代码。使用它的唯一限制是:由 GTK + 库衍生的库同样必须受开放源代码的许可证条款的制约

GTK + 使用的许可证是重要的。由于提供源代码,所以只要有开发人员在使用它,这一产品就不可能消失。由于提供源代码,任何开发人员都可能排除他或她手头使用的版本的隐错;将排除隐错的结果提交 GTK + 的维护人员,并由他们将改正的结果吸收到它的主要发行版中。许多使用 GTK + 的开发人员以这种方式成为 GTK + 的开发人员,同时进一步提高了

GTK+的稳定性和性能。

1.3 从何处得到 GTK+

可以从 GTK+的 Web 站点(www.gtk.org)下载它。它使用的版本号类似于 Linux 使用的版本号,偶数(如 1.0 和 1.2)表示稳定的版本;奇数(如 .99 和 1.1)表示开发版本。有时还增加一个附加数字表示对这一版本进行了隐错的改正。

1.4 如何建立 GTK+

为了建立 GTK+,首先需要从 GTK+的 Web 站点下载 GLIB 和 GTK+。因为在 GTK+内部许多地方用到 GLIB,需要先建立 GLIB。在建立 GLIB 和安装它以后,就可以建立GTK+了。

1.5 如何从源代码建立 GTK+

作为新手,如何从源代码建立 GTK+呢?可以用几个简单步骤来建立 GLIB 和 GTK+软件包。下载的文件的名称类似 GTK+-1.2.0.tar.gz。因为它的后缀是 .tar.gz,这是归档和经过压缩的文件。所以必须在命令行使用 gunzip 命令对这一文件进行解压缩:

```
gunzip GTK+-1.2.0.tar.gz
```

用 gunzip 解压缩后得到容量比较大的 tar 文件,一般具有类似 GTK+-1.2.0.tar 的名称。现在用下面的 tar 命令将它扩展为它的目录结构:

```
tar xvf GTK+-1.2.0.tar
```

这一命令建立建库所需要的目录结构。进入刚才建立的目录,并敲入如下的命令:

```
./configure
```

用这种方式运行配置实用程序(configure),用以建立建库所需要的 makefile。如果不出错误,可以敲入下列命令来建库:

```
make
```

建库后,需要安装库。但是,必须作为 root 用户或者作为具有合适权限的用户来安装库。在系统上用来安装库的命令是:

```
make install
```

用户可能需要在自己的系统上运行/sbin/ldconfig 使 GTK+能正常工作。当然,需要在建立 GTK+以前先建立和安装 GLIB。

1.6　运行实例需要什么条件

这些实例曾经在 Linux 2.0 上用 gcc 2.7.2 编译软件编译过。虽然尚未对 egcs 编译软件进行过测试，但它应该能用在这些实例上。这些实例用的是 GTK + 1.1.5，它们应该能在 GTK + 1.2 上正常进行工作。不对源代码进行一定的修改，许多实例不能在 GTK + 1.0.x 上工作。假定一切配置正确，应该能敲入 make 来建立程序。

1.7　从何处得到样本程序

对那些不想花长时间敲入实例的人们，可以从下列 URL 上下载这些程序：

http://www.mcp.com/product-support/

在它提供的对话框中敲入本书的 ISBN(0735700214)，就可以得到建立实例所需要的全部源代码。运行实例需要 GTK + 1.2。

1.8　从何处可以得到更多的帮助

除了阅读本书外，还可以转向其他一些地方取得帮助。GTK + 的 Web 站点提供可以用来学习 GTK + 的一本教材，若干 GTK + 的样本程序，以及其他 GTK + 应用程序的资源库。在与 GTK + 源代码一起提供的 examples 目录中包含不同的样本应用程序。与 GTK + 源代码一起提供的 testGTK + 应用程序可以用来测试 GTK + 的每一种构件(widget)。当不能发现特定构件的文档时，testGTK + 应用程序的源代码是有价值的，因为它的代码可能就在 testGTK + 应用程序中。对有志进取的人们来说，GTK + 的源代码总是可以得到的。

如果你不能从上述信息源得到问题的答复，可以试试各种针对 GTK + 的邮递表。许多知识丰富的 GTK + 开发人员通过邮递表回答从简单的到非常复杂的问题。主要的邮递表有：GTK-list@redhat.com 和 GTK-app-devel-list@redhat.com。要订阅 GTK-list，用“subscribe”为主题向 GTK-list-request@redhat.com 发送电子邮件。与此类似，要订阅 GTK-app-devel-list，用“subscribe”为主题向 GTK-app-devel-list-request@redhat.com 发送电子邮件。这些邮递表(特别是 GTK-list)的通信量比较大，大约每天有 50 份电子邮件。有些天甚至比这一数字还要高。

第 2 章　GLIB

GLIB 库是由 GTK + 频繁使用的流行函数的集合。链接表、树、错误处理、内存管理和计时器只是库的内容的一部分。本章讨论在 GLIB 库中最常用到的函数。GTK + 需要 GLIB 库，它的可移植性和功能都依赖于 GLIB 库。GLIB 库可以单独使用；可以用它来开发非图形用户界面(non-GNU)应用软件。

2.1　类型

GLIB 用它自己的一组类型(type)，而不用在 C 编程语言中使用的标准类型。这种做法使移植到其他平台的工作更为方便，允许改变数据类型而不必重写应用软件。由于 GTK + 使用 GLIB 的数据类型和函数，从理论上讲，移植 GTK + 需要将 GTK + 使用的 GLIB 以及 GDK 库移植到目标平台。在不同平台之间移植应用软件需要巨大的耐心和可移植设计，特别是在不相似的平台之间移植软件更是如此(如 Linux 到 Windows 的移植就比 Linux 到 UNIX 的移植要复杂)。

GLIB 使用的许多类型稍稍不同于标准 C 的数据类型。如在 C 中的 char 数据类型在 GLIB 中变成 gchar 数据类型。虽然在 Intel 平台的 GLIB 头文件中 gchar 可能被定义为 char。为了保持可移植性，两者之间不可以交换使用。许多函数使用 gchar * 代替 char * 作为参数。这种改变虽不重要，但需要习惯使用它。一些常用的数据类型的对应关系如下：

C 类型	GLIB 类型
char	gchar
short	gshort
long	glong
int	gint
char	gboolean
viod *	gpointer

在 GLIB/GTK + 的应用软件中使用 GLIB 数据类型保证：当实施基础的数据类型(如 gboolean)改变时应用软件仍能正常工作。例如，在以后的发行版中可以将 gboolean 数据类型定义为 int；使用 gboolean 数据类型代替 char，保证仍能对应用软件进行正常编译。

2.2　消息

GLIB 用 4 种函数来显示消息。对 GUI/non-GUI 开发来说，每种函数都能在运行中(on-the-fiy)进行扩展。从显示不可恢复的错误消息的 g_error 到标准的输出函数 g_print，这 4 种不同函数有 4 种层次的消息处理。每种消息函数显示不同类型的消息，并和 printf 函数一

样能够传递可变数目的参数。

2.2.1　*g_error*

g_error 函数显示应用软件中的严重错误。它显示错误消息，并结束程序的运行。只当可能发生导致程序退出的错误时才使用这一函数。函数 g_set_error_handler 可以覆盖 g_error 的功能，但它不能阻止 g_error 函数结束程序的运行

2.2.2　*g_warning*

当发生可恢复的错误而且程序能够继续运行时，函数 g_warning 显示错误消息。GTK+ 用这一函数向程序设计人员显示已经成功处理了的错误。g_set_warning_handler 函数可以撤消 g_warning 的缺省功能。

2.2.3　*g_message*

函数 g_message 显示与错误无关的消息。函数 g_set_message_handler 能够撤消 g_message 函数的功能。

2.2.4　*g_print*

g_print 函数通常用于调试。你可以在开发过程中使用 g_print 函数，并在实际使用时撤消它的功能，使它不显示任何消息。这种方法快而方便，而且快于从代码中移去调试工具。你可以使用 g_set_print_handler 函数撤消 g_print 函数的功能。

你可以用传递新的处理函数的方法来撤消每个消息子程序。新的消息处理程序可以弹出带有正文"log the message/error to a file(将消息/错误记录到一个文件)"的窗口，或者完成其他的操作。在开发过程中，可以用这种方法很快改变消息函数的功能以适应开发人员的需要。

2.2.5　定制错误处理

这一简单的例子说明如何定制错误处理。传递 normal 参数将使用 GLIB 提供的缺省消息。

```
[brain@sharp messages]$ message normal
He's a print
message: Here's a message

** WARNING **: Here's a warning

** ERROR **: Here's an error

Aborted (core dumped)
```

给显示缺省消息的同一个程序传递参数 surfer，现在它将以稍稍不同的方式显示同一些

消息。要注意:虽然安装了定制错误处理程序,当发生错误时仍然结束程序的运行。

```
[brain@sharp messages]$ message surfer
Dude, Here's a print
Dude, ya got a message - Here's a message
Bad news Dude. - Here's a warning
Major wipe out, dude. - Here's an error
Aborted (core dumped)
[brain@sharp messages]$ exit
```

下面是说明程序的代码。当传递 surfer 参数时,在调用消息函数以前,设置了新的消息处理程序。

```
/*
 * Example showing message behavior
 */
#include <glib.h>

/*
 * Surfer Print
 *
 * Override function for the g_print function.
 */
void SurferPrint (const gchar *buf)
{
  printf ("Dude, ");
  printf (buf);
}
/*
 * Surfer Message
 *
 * Override for the g_message function
 */
void SurferMessage (const gchar *buf)
{
  printf ("Dude, ya got a message - ");
  printf (buf);
}
/*
 * SurferWarning
 *
 * Override for the g_warning function
 */
```

```
void SurferWarning (const gchar * buf)
{
  printf ("Bad news Dude. - ");
  printf (buf);
}
/*
 * SurferError
 *
 * Override for the g_error function
 */
void SurferError (const gchar * buf)
{
  printf ("Major wipe out, dude. - ");
  printf (buf);
}
/*
 * ShowParams
 *
 * Show the options available for running the program.
 */
void ShowParams ()
{
  printf ("Must pass in parameter. Valid parameters are: \n");
  printf (" 'surfer' - use surfer message handling. \n");
  printf (" 'normal' - use normal message handling. \n");
  exit (0);
}
/*
 * main
 *
 * Program begins here
 */
int main (int argc, char * argv[])
{
    /* --- Not enough args? --- */
    if (argc <= 1) {
        ShowParams ();
    }

    /* --- Normal speech? --- */
    if (strcmp (argv[1], "normal") == 0) {
        /* --- Do nothing - just verify that parameter is valid --- */
```

```
/* --- Surfer speech? --- */
} else if (strcmp (argv[1], "surfer") == 0) {
      /* --- Seems that they want surfer speech for the errors. --- */
      g_set_error_handler (SurferError);
      g_set_warning_handler (SurferWarning);
      g_set_message_handler (SurferMessage);
      g_set_print_handler (SurferPrint);
} else {
    /* --- Can only pick 'normal' or 'surfer' --- */
    ShowParams ();
}

/*
 * Show functions at work. If we have custom handlers,
 * the message will be intercepted.
 */
g_print ("Here's a print \n");
g_message ("Here's a message \n");
g_warning ("Here's a warning \n");
/*      This terminates the program - always --- */
g_error ("Here's an error \n");

}
```

2.3 判断

为验证在代码中所作的假定,在开发过程在中需要使用判断。如果判断失败,可能导致严重问题。用 g_assert 函数来检查判断。例如,如果将函数指针传递给函数,而且它应该有它的值;你就可以用 g_assert 函数验证这一假定。如果判断失败,程序退出。并显示错误消息和判断失败的行号。

```
g_assert (ptr != NULL)
```

判断应该只用在条件本来不应该发生的场合。判断通常用在全面进行错误处理以前。例如,在开发周期的开始阶段,在代码中增加判断要比加入错误检查和进行相应的错误处理更为方便。在发行软件时,应该用更为优雅的处理错误条件的方式代替判断。在发行的代码中只能在有限范围内使用判断,而且只用于本来不应该发生的条件。如果发生这样的条件,表示问题严重。

你可以在代码段中不应该到达的位置插入 g_assert_not_reached 函数,如果在程序执行过程中到达这一位置,程序停止并显示错误消息。

通过在编译时定义 G_DISABLE_ASSERT,可以将两种判断函数从正式使用的程序中移去。

2.4　字符串函数

GLIB使用它自己的 GString 数据类型代替 char 和 gchar 数据类型来处理字符串。GString最好的地方是它为你处理内存分配。通过使用 g_string_new 函数和传递字符串来建立 GString:

```
gStr = g_string_new("Hello");
```

实际上,GString是包含有关字符串信息(称为 str 的 gchar *)和长度(len)的结构。为了检查,可以从结构访问字符串和它的长度;但是任何这样的操作都应该通过 wrapper 类来完成。需要使用 g_string_free 函数释放已经建立的字符串结构;它取字符串并释放分配给它的内存。g_string_free 函数取第二个参数,用它来确定是否应该释放 GString 的字符串内存。在通常情况下,由于 GString 函数分配了内存,这一参数应设置为 TRUE,并在释放数据结构后应释放分配的内存。如果在其他位置引用了 GString 的字符串内存(char *),它应该被设置为 FALSE。

```
g_string_free(gStr, TRUE);
```

可以使用 g_string_assign 函数给已有的 GString 赋新值。它用新值代替已有的值,并处理新的内存分配。

```
g_string_assign(gStr, "New String Value");
```

使用 g_string_truncate 函数,可以将字符串截为特定的长度。传递 0 作为新的字符串长度等效于将字符串设置为空("")。

```
g_string_truncate(gStr, 0);
```

使用 g_string_append 函数可以将字符串接在已有的 GString 字符串的后面。由函数自动完成附加内存的分配。

```
g_string_append(gStr, "Adding more");
```

还可以使用 g_string_prepend 函数将字符串插在 GString 的开始。使用的参数和 g_string_append 函数使用的相同。但是它将字符串插在已有的字符串前面。

```
g_string_prepend(gStr, "Adding before");
```

使用 g_string_append_c 或 g_string_prepend_c 函数可以将字符插在 GString 的后面或前面。

```
/* ---Append a 'z' to the end--- */
g_string_append_c(gStr,'z');
```

```
/* --- Put an 'a' on the beginning --- */
g_string_prepend_c (gStr, 'a');
```

使用 g_string_up 函数可以将字符串转换为大写;使用 g_string_down 函数可以将字符串转换为小写。这两个函数转换 GString,并用转换结果代替原来的字符串。

```
/* --- Convert the string to upper case--- */
g_string_up (gStr);

/* --- Convert the string to lower case --- */
g_string_down (gStr);
```

2.5 单链接表

GLIB 库具有在链接表中存储数据的有用函数。链接表的数据类型是 GSList,修改链接表(增加或删除链接表表项)的函数返回 GSList 指针。GSList 的数据结构由两部分组成。分别定义如下:

```
gpointer data;
GSList *next;
```

在这一结构中的 data 字段存储数据,它通常是指向另一个结构的指针。它的 next 字段是指向链接表的下一个表元的指针。图 2.1 说明这种安排。

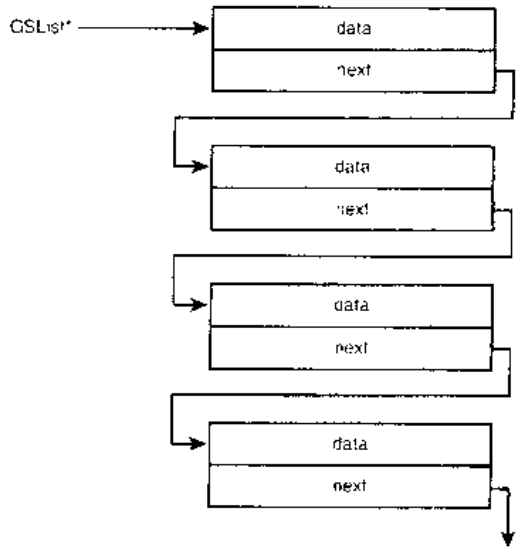


图 2.1 GSList

在下面的例子中传递一字符串。事实上,存储在链接表中的数据可以是任何类型的数据。必须经常将 GSList * 初始化为 NULL。不这样做会出问题。因为非空(non-NULL)指针表示一个链接表,会导致内存问题或者程序的非正常结束。

2.5.1 增加表项

在链接表中增加表项的一种方法是利用 g_slist_append 函数建立结点(node),并将它加到链接表的最后。例如,可以按如下方式将“Wilma”加到链接表的最后:

```
GSList *list = NULL;

sName = "Wilma";
list = g_slist_append(list, sName);
```

或者:

```
list = g_slist_append(list, "Wilma");
```

由 g_slist_append 函数返回的值是新的链接表。当插入或删除表项的操作牵涉到表头改变时,将返回新表。可以用 g_slist_prepend 函数将“joe”加到表的开始:

```
list = g_slist_prepend(list, "joe");
```

可以使用 g_slist_insert 函数将表项插入表中的特定位置,并提供位置索引以插入表项。例如,可以在表的第一个表项后面插入“Mary”:

```
list = g_slist_insert(list, "Mary", 1);
```

如果不指定插入位置,则将表项加到表的最后。

链接表子程序并不知道保存在表的每一个结点的数据类型。如果需要时,开发人员必须分配和释放保存数据的内存。

2.5.2 保持链接表的排序

由于表的数据项可以表示任何类型的数据,而且链接表并不知道数据的具体结构,试图保持链接表的排序是困难的。所幸的是,可以用比较函数将表项加到排序的链接表中。比较函数取两个数据项,并对它们进行比较。如果第一项大于第二项,返回 1 值;如果第一项等于第二项,返回 0 值;如果第一项小于第二项,返回 -1 值。你可以建立简单的比较函数对字符串数据项进行比较和对它们进行排序:

```
gint CompareNames (gconstpointer sName1, gconstpointer sName2)
{
  return ((gint strcmp (char *) sName1, (char *) sName2));
}
```

有了比较函数以后,可以使用 g_slist_insert_sorted 函数和比较函数将表项插入排序的

链接表。调用比较函数时用进行比较的链接表的表项和被插入的数据项作为参数:

```
list = g_slist_insert_sorted(list,
                             "Anne",
                             CompareNames);
```

2.5.3 查找链接表的表项

使用 g_slist_find 函数查找链接表中的数据项。如果在链接表中没有找到数据,则返回 NULL 值。

```
item = g_slist_find(list, "Joe");
```

函数 g_slist_find 并不确切知道结点的数据项。因此,它比较数据项的值;通常它们是一些指针。在下列情况下,这种技术可能出问题。

```
char szMary[20];

/* --- Put "Mary" into a string --- */
strcpy(szMary, "Mary");

/* --- Add "Mary" to the list --- */
list = g_slist_append(list, "Mary");

/* --- Look for "Mary" from the string --- */
item = g_slist_find(list, szMary);

/* --- Whaddya mean it's not found? I just added it. --- */
```

在这种情况下,由于 szMarys 数组指向错误的内存地址,所以没有在表中找到"Mary"。

2.5.4 链接表的长度

链接表的长度可以用 g_slist_length 函数进行计算。

```
guint nLenth = g_slist_length(list);
```

2.5.5 从链接表删除表项

使用 g_slist_remove 函数可以从表中删除表项。如果在表中没有这一表项,不进行任何操作。而且也不给你提供错误或警告消息。如果数据项是一个指针,它必须与被删除的项匹配。你可以试用下列方式删除数据项:

```
list = g_slist_remove(list, data);
```

与 g_slist_find 函数一样,g_slist_remove 函数只对数据项进行比较。如果数据项是指

针，它就不知道数据是什么。在数据项中存储整数或简单的数据类型不成问题，如果存储的是指针，就只进行指针的比较（data == node->data），并不对地址的内容进行比较。如果你用链接表存储字符串，了解这一点是重要的。此外，如果开发人员分配内存，删除表项后应该释放内存。

2.5.6　取得第 n 个表项

通过传递你需要的表项的索引，可以使用 g_slist_nth 函数查找链接表中的第 n 个表项。下面的例子取得表中第 7 个表项：

```
node = g_slist_nth (list, 7);
```

可以使用 g_slist_index 函数查找数据项的索引。由于这一函数调用 g_slist_find 函数，在“2.5.3　查找链接表的表项”中提到的警告也适合这一函数。例如：

```
nIndex = g_slist_index (list, 22);
```

2.5.7　查看链接表

第一种查看链接表表项的方法是在循环中自上而下地检查表项。使用 node->next 取得链接表中下一个结点。

```
/* --- List is the head --- */

/* --- Loop through the list --- */
for (node = list; node; node = node->next) {

  /* --- Dump out the contents, assumes data is char --- */
  g_print ("%s \n", (char *) node->data);
}
```

第二种方法是使用 g_slist_foreach 函数。函数 g_slist_foreach 调用另一个函数，并从链接表的每一个结点将数据项传递给这一函数。为了使用 g_slist_foreach 函数，你首先需要建立函数 PrintNames 来正确地处理数据项。PrintNames 函数取得数据，并将它显示出来。PrintNames 函数有两个参数。其中 data 参数是被显示的数据项，它是从链接表的每个表项传递过来的；而 user_data 是传递给每个数据项的附加信息。

```
void PrintNames (gpointer data, gpointer user_data)
{
   gchar   *message;

   /* --- Convert the data to a string --- */
   message = (gchar *) data;
   /* --- Display the string --- */
   g_print ("%s\n", message);
```

```
}
```

虽然这一函数只显示一个数据项,但是 g_slist_foreach 函数对链接表的每个数据项调用它一次。可以按如下方式设置 g_slist_foreach 函数,使它调用 PrintNames 函数。

```
/* --- Another way to print each of the data elements --- */
/* --- Call PrintNames for each element,user_data = NULL---*/
g_slist_foreach (list, (GFunc) PrintNames, NULL);
```

在本例中,PrintNames 函数的 user_data 参数对链接表的每个数据项传递 NULL 值。可以容易地将这一参数改变为其他信息供 PrintNames 函数使用。

2.5.8 释放链接表

为了释放整个链接表,可以使用 g_slist_free 函数。按如下方式将被释放的链接表的名称传递给函数:

```
g_slist_free (list);
```

如果数据项(的内容)是分配内存的指针,应该在调用 g_slist_free 函数前先释放内存;否则将造成内存的漏损。

2.6 双链接表

GLIB 还有一组使用 GList 数据类型的双链接表函数。这些函数在操作上类似于单链接

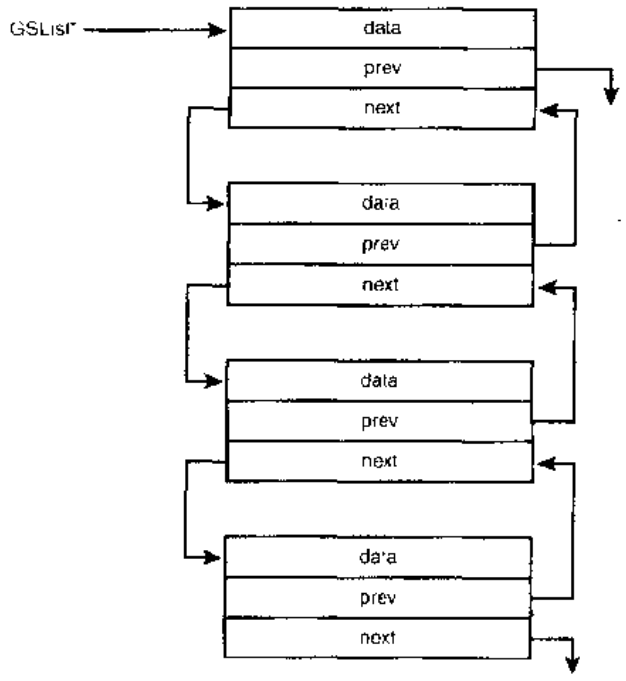


图 2.2 GList

表函数，但双链接表函数的前缀用 g_list 代替 g_slist；而且，双链接表函数使用 GList 数据类型代替 GSList 数据类型。GList 数据类型同时具有到前一个数据项和到下一个数据项的链接，它使反向查找非常方便（见图 2.2）。

2.6.1　链接表的性能

链接表经常用于堆栈（在表的前端增加和移去数据项）和小表；用它保存大量数据时它的代价高昂。当用链接表来保存信息时，应该将数据加在表的前端；这样做可以很快插入表中。经常从表中搜索数据可能代价高昂，而且与表长度有关。

2.7　Hash 表

GLIB 具有一组用于 Hash（散列）表的函数。Hash 表是存储和查找信息的快速方法。数据项用关键字（key）进行存储，关键字同样用来检索数据项的值。图 2.3 说明如何存储 Hash 表的值。

Hash 表需要用回呼函数（callback function）来计算 hash 值，而且回呼函数必须正确计算出唯一的值。

2.7.1　建立 Hash 表

可以使用 g_hash_table_new 函数建立 Hash 表，它返回指向 GHashTable（Hash 表）的指针。这一函数需要一个 hash 函数和一个比较函数。它使用 hash 函数计算关键字的 hash 值，并用 hash 值来确定将关键字存储在哪一个容器（bucket）中。在查找数据项时，用比较函数比较关键字。如果将字符串数据加到 Hash 表中，可以用一个简单函数；它取字符串的前两个字符来计算 hash 值，建立 Hash 表，并将字符串存储在 Hash 表中。

```
/*
 * Make a hash from the first two characters.
 */
guint HashFunction (gpointer key)
{
  char *sKey;

    sKey = key;
    return (sKey[0] + sKey[1]);
  }
```

然而，本例不是好的 hash 函数；因为这一函数只用字符串的前两个字符来计算 hasb 值，像 linux、linus、like 和 list 这样一些词最终都得到相同的 hash 值。更好的 hash 函数将充分利用数据本身来得到更多唯一的关键字。下面的 hash 函数使用整个字符串来计算 hasb 值。这一函数的运行时间会比前面的例子长。

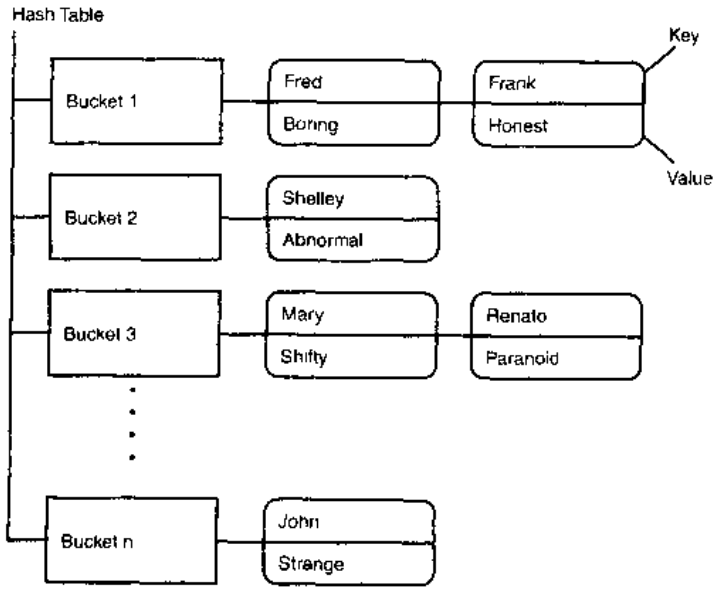


图 2.3 Hash 表

```
guint HashFunction (gpointer key)
{
   char * sKey;
   guint giHashValue = 0;
   int nIndex;

   sKey = skey;

   /* --- Some error handling --- */
   if (key == NULL) return (0);
   /* --- Compute hash based on entire string --- */
   for (nIndex = 0; nIndex < strlen (sKey); nIndex++ ) {

      /* --- Shift left because sequence is important --- */
      giHashValue = (giHashValue << 4) +
                    (giHashValue ^ (guint) sKey[nIndex]);
   }

   return (giHashValue);
}
```

Hash 表的字符串比较函数只用字符串比较来确定两个字符串是否相等。这一步是需要的,因为 hash 函数可能对两个不同的字符串返回相同的 hash 关键字。字符串 hash 比较函数类似下面的例子:

```
gint HashCompare (gpointer sName1, gpointer sName2)
```

```
{
    return (!strcmp ((char *) sName1, (char *) sName2));
}
```

有了这两个函数以后,就可以用这两个函数名称作为参数调用 g_hash_table_new 函数来建立 hash 表。

```
hTable = g_hash_table_new (HashFunction, HashCompare));
```

建立 hash 表后,可以将数据项存储到 hash 表中。函数 g_hash_table_insert 将数据项存储到表中。存储数据项时,hash 表取 3 个参数:用 g_hash_table_new 函数建立的 hash 表、使数据项可以被查找的关键字以及与关键字结合的值。下列代码将数据项存储在 hash 表中:

```
g_hash_table_insert (hTable,"Fred","Boring");
```

在上面的例子中,将值"Boring"和关键字"Fred"存储在表中。函数 g_hash_tahle_lookup 用来查找关键字的值。对关键字"Fred"将返回值"Boring"值。

```
sValue = g_hash_table_lookup (hTable,"Fred");
```

类似于普通表,Hash 表也有 foreach 函数;它可以用来显示 hash 表中的所有数据项。你需要建立回呼函数来显示信息。它稍稍不同于链接表的回呼函数,因为 hash 表的回呼函数需要同时传递关键字和值。

```
void PrintNames (gpointer key, gpointer value, gpointer user_data)
{
   g_print ("Key: %s, Value: %s \n",
           (gchar *) key, (gchar *) value);
}
```

然后,可以使用 PrintNames 函数名称作为参数调用 g_hash_table_foreach 函数:

```
g_hash_table_foreach (hTable, (GHFunc) PrintNames, NULL);
```

可以使用在 g_hash_table_foreach 函数中的第三个参数给 PrintNames 函数传递附加信息,它将作为回呼函数的用户数据变量。但是,使用 g_hash_tahle_foreach 函数时数据项出现的顺序是随机的。

2.7.2　从 Hash 表删除数据项

可以调用 g_hash_table_remove 函数删除 hash 表中的数据项。调用时,提供需要从表中删除的数据项的关键字作为参数。

```
g_hash_table_remove (hTable, "Fred");
```

2.7.3 删除 Hash 表

在用完 Hash 表后,应该用 g_hash_table_destroy 函数删除 Hash 表。和其他删除函数一样,g_hash_table_destroy 函数只删除 Hash 表,并不删除用户分配的用在 Hash 表中存储数据的内存。

```
g_hash_table_destroy (hTable);
```

2.8 树

树是存储信息的大型数据结构。从内部看树要比链接表和 Hash 表复杂。但从树中检索信息的速度要比链接表快。而且不像 Hash 表,树可以存储排序的信息。

2.8.1 比较函数

用 g_tree_new 函数建立树,它返回指向 Gtree(树)的指针。这一函数要求用比较函数作为参数,使树知道如何调整它本身以达到快的访问信息的速度。如果你存储字符串,可以使用类似如下的比较函数:

```
gint CompareNames (gpointer name1, gpointer name2)
{
   return (strcmp (name1, name2));
}
```

2.8.2 建立树

在建立比较函数后,可以将比较函数的名称作为参数传递给 g_tree_new 函数,使它对树进行初始化。

```
tree = g_tree_new (CompareNames);
```

2.8.3 插入数据项

在建立树以后,可以将数据存储到树中。比较函数使保存在树中的数据按排序的方式保存。可以使用 g_tree_insert 函数将数据存储在树中。这一函数要求以关键字/值对的组合形式提供数据。例如,你可以将关键字 "Fred" 和它的值"Loud"按如下方式存储到树中:

```
g_tree_insert (tree, "Fred", "Loud");
```

2.8.4 查找数据项

用函数 g_tree_lookup 查找在树中的值。将关键字传递给 g_tree_lookup 函数作为参数。如果能在树中找到它的对应值,它将返回关键字的对应值。为了查找刚才保存的的关键

字“Fred”的值，可按如下方式调用 g_tree_lookup 函数：

```
sValue = g_tree_lookup (tree, "Fred");
```

上面的语句返回“Loud”值，因为它是和关键字“Fred”一起存储的。

2.8.5　遍历树

正如 Hash 表和链接表类型具有 foreach 函数遍历所有的数据项一样，树也有遍历树的函数。在这种情况下，函数比较复杂。函数 g_tree_traverse 有一个附加参数，用来确定如何穿越树。树可以有 3 种方式遍历：顺序遍历(G_IN_ORDER)、先根遍历(G_PRE_ORDER)和后根遍历(G_POST_ORDER)。图 2.4、2.5 和 2.6 分别表示 3 种不同的遍历方式

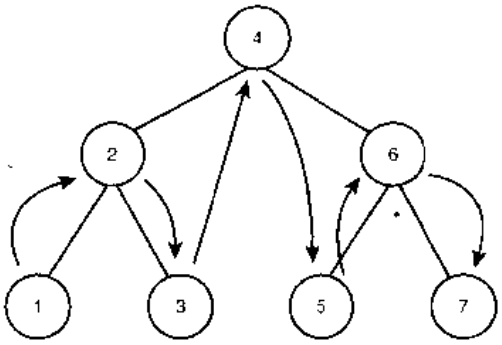


图 2.4　顺序遍历

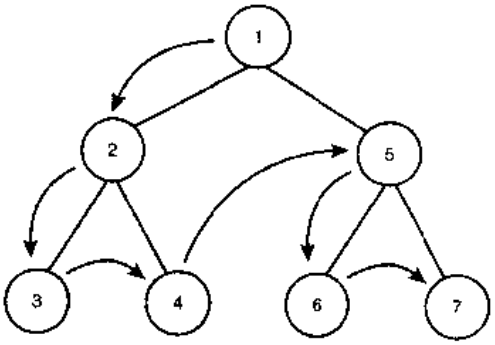


图 2.5　先根遍历

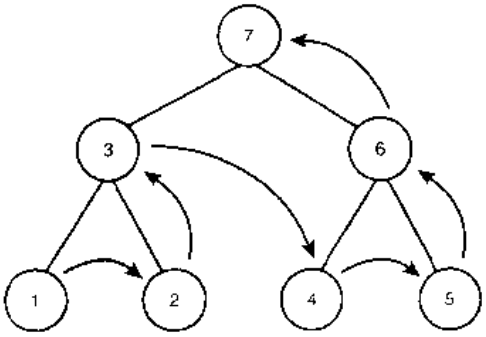


图 2.6　后根遍历

按字母顺序遍历树，使用 G_IN_ORDER 遍历。下面是调用函数的例子：

```
g_tree_traverse (tree, TraverseTree, G_IN_ORDER, NULL);
```

当然，你需要用 TraverseTree 函数显示在树中的每一对关键字和值。

```
gint TraverseTree (gpointer key, gpointer value, gpointer data)
{
```

```
    char * sKey = key;
    char * sValue = value;
    g_print ("key: %s, Value: %s", sKey, sValue);
    return FALSE;
}
```

2.9 内存管理

GLIB 库提供一些内存管理函数来代替标准的 malloc/free 函数。保持 GLIB 库的命名标准,函数 malloc 的代用函数称为 g_malloc。与 malloc 函数一样,它使用的参数是请求的内存块的容量。GLIB 库还提供 g_free 函数代替标准的 free 函数,以及 g_realloc 函数代替标准的 realloc 函数。

```
/* --- Allocate some memory --- */
ptr = g_malloc (10 * sizeof (char));

/* --- Expand the memory --- */
ptr = g_realloc (ptr, 20 * sizeof (char));

/* --- Free the memory --- */
g_free (ptr);
```

GLIB 内存函数提供跟踪内存问题的能力。当编译 GTK + /GLIB 时,你可以在 GLIB/gmem.c 中定义 MEM_CHECK 和/或 MEM_PROFILE,将比较慢但更有帮助的功能编译到内存管理函数中。

如果在编译 GLIB 时定义了 MEM_PROFILE,GLIB 将保持跟踪通过 GLIB 的内存分配。这将简化跟踪内存漏损问题。调用 g_mem_profile 将显示类似如下的有关内存的信息:

```
1 allocation of 8 bytes
1 allocation of 52 bytes
2 allocation of 1024 bytes
2108 bytes allocted
0 bytes freed
2108 bytes in use
```

在 GLIB 中使用 MEM_CHECK 将跟踪已经释放的指针。试图多次释放同一个指针将导致下列错误:

```
* * WARNING * *: freeing previously freed memory
```

这种错误信息表示你的指针管理并不像你想象的那么周到。

2.10　小结

GLIB 能够处理许多常用的数据结构，包括链接表、Hash 表和树，并提供一组标准的消息函数。这些函数由 GTK + 库所使用。GLIB 还提供可用于所有平台的一组标准子程序。你可以与 GTK + 一起使用 GLIB，也可以单独使用 GLIB。

第3章　开发GUI应用

在Linux系统上，缺乏可供自由使用的单一的开发图形用户界面(GUI)应用的标准。除非把X Window库计算在内，但它是非常低层的库，不适合快速开发应用。在Internet档案中，充满了简化应用开发的GUI工具集；但没有一种工具集吸引大量的追随者。Linux的GUI团体因而解体。当TrollTech(www.troll.no)开发了称为Qt的优秀的GUI工具集后(它非常适合于开发Linux的GUI应用)，情况开始发生变化。然而，TrollTech的许可证条款的限制妨碍了许多开发人员使用它的产品。Qt的许可证条款指出：除非开发人员购买了Qt库的商业版本，否则在Qt库基础上开发的应用软件必须是自由的。本人认为：对Qt库的这种限制，妨碍了商业公司利用优秀的工具集开发Linux的应用。尽管如此，Qt库对非商业应用是自由的。

Linux的发行受到GNU通用公共许可证(GPL，GNU General Public Licence)条款的制约。它允许任何人取得和修改源代码，并重新发布源代码。因此，许多开发人员认为任何用于开发Linux应用的工具集应该也受到GPL条款的制约。这种方式既降低了对商业公司改正软件隐错的依赖，又不必担心开发人员过分受单一实体的控制。这种情况导致了GIMP工具集(GTK+)的开发。

GIMP工具集(GTK+)是最新用于Linux的GPL GUI库。它对商业和非商业开发都是自由的。GTK+也是跨平台的工具集。它在许多UNIX机器上和微软的Windows上都得到支持。由于得到一些较大的Linux软件发行公司的支持，并有一大批Linux开发人员作后盾，GTK+已经演变为非常优秀的开发GUI应用的工具集。

GTK+名称的由来是由于人们利用这个库建立了GIMP(GNU Image Manipulation Project)。GIMP是用GTK+写成的具有商业软件质量的卓越的图形编辑软件。GIMP类似于Photoshop，它可以从www.gimp.org站点得到。

GTK+是用C语言写成的，而且是面向对象的。它提供了从多种语言使用GTK+的接口(binding)，包括C++(很奇怪，它使用了称为GTK--的包装程序)。类似大多数现代的GUI工具集，GTK+也是由事件驱动的工具集。屏幕用构件(widget)建立，如窗口、组合框、正文框和按钮等。在这些构件中设置回呼函数来完成对信号的处理。这些信号通常是鼠标或键盘产生的事件引起的。当回呼函数收到信号时，应用程序响应信号，并完成某些处理。信号的概念类似在java或Windows中使用的事件的概念。

GTK+是建立在GDK(GIMP Drawing Kit)基础之上的构件库。GDK是Xlib的包装程序(wrapper)。GTK+在显示构件时调用GDK。GDK是依赖于平台的应用编程界面(API)。它建立在特定平台原有的API之上(如Xlib，Win32)，并提供图形API。因为GDK位于GTK+和Xlib之间，只要改变GDK，就可以将GTK+移植到其他操作系统上(见图3.1)。GTK+和GDK都依赖于GLIB，后者提供许多处理常用数据类型(如表、树和字符串)的函数，内存分配函数和错误处理子程序。可以不必通过GTK+直接调用GDK和GLIB。

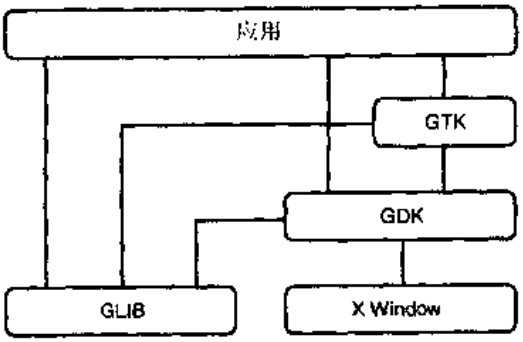


图 3.1　GTK + 的应用层

3.1　建立 GTK + 应用

为了建立 GTK + 应用，在应用程序文件中用到 GTK + 函数调用或定义的每一部分必须包含 gtk/gtk.h 文件。后者是 GTK + 的主要包含文件。此外，还必须连接若干库。所幸的是，GTK + 的开发人员为你提供了方便。使用 gtk-config 程序，可以简化这些工作。你可以用下列命令行编译简单的 GTK + 程序：

```
gcc -Wall -g example.c -o example 'gtk-config -cflags' 'gtk-config -libs'
```

在 ('gtk-config -cflags'和'gtk-config -libs' 两边的引号是单反引号。在本例中，example.c 是被编译的程序。而'gtk-config -cflags'实际运行了 gtk-config 程序，它输出编译标志，并将它们插入命令行。用'gtk-config -libs'输出连接标志并插入命令行。在 GTK + 中包含 gtk-config 程序。在 Linux 的命令行提示符后敲入 gtk-config -cflags 和 gtk-config -libs 表示给 GNU C 编译软件(gcc)传递参数的正确方法。

3.1.1　初始化 GTK +

写 GTK + 程序需要调用 gtk _ init 函数对 GTK + 库进行初始化。将应用程序的变元(argc,argv)传递给 gtk _ init 函数，并检查主要用于调试的 GTK + 选项。如果在变元列表中出现任何这样的 GTK + 参数，就将它移去。在运行 gtk _ init 函数后，它们不应该出现在应用程序中。

```
/* --- Initialize GTK --- */
gtk_ init (&argc, &argv);
```

对 GTK + 进行初始化后，就可以调用其他GTK + 库函数。

3.1.2　建立窗口

GTK + 的构件(widget)是 GUI 的组成部分。窗口、检查框、按钮和编辑字段都属于构件。通常将构件和窗口定义为指向 GtkWidget 结构的指针。在 GTK + 中，GtkWidget 是用于所有

构件和窗口的通用数据类型。

在对 GTK + 库进行初始化后,大多数应用建立一个主窗口。在 GTK + 中,主窗口常常被称为顶层窗口(top-level window)。顶层窗口不被包含在任何其他窗口内,所以它没有上层窗口。在 GTK + 中,构件具有所谓的父/子关系,其中父构件是容器,而子构件则是包含在容器中的构件。顶层窗口没有父窗口,但可能成为其他构件的容器。

在 GTK + 中建立构件分两步:建立构件,然后使它可以看得见。可以用这种技术建立顶层窗口。调用带 GTK _ WINDOW _ TOPLEVEL 参数的 gtk _ window _ new 函数建立新的顶层窗口。gtk _ window _ new 函数返回指向 GtkWidget 结构的指针。刚建立的窗口是看不见的,必须调用 gtk _ widget _ show 函数使它成为可见。

```
/* --- Create a top-level window in gtk --- */
/* --- note the window is NOT visiable yet --- */
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

/* --- Now, make the window visiable --- */
gtk_widget_show (window);
```

3.1.3 GTK + 的事件循环

对 GTK + 进行初始化并将窗口和构件置于屏幕以后,应用软件使 GTK + 开始执行,所以它可以对事件(如鼠标的移动、按键等)进行处理。但是,在调用 gtk _ main _ quit 函数以前,对 gtk _ main 函数的调用并不返回。应用程序如何才能调用 gtk _ main _ quit 函数? 在这种情况下,在调用 gtk _ main 函数以前应该建立回呼函数。而且设置 GTK + 使得在出现某些信号时转而执行应用软件,由应用软件接着进行处理。

GTK + 程序可以非常小。例如,下面的基本程序用来说明 GTK + 程序的流程。这一程序没有回呼函数,所以必须人工撤消它。

```
#include <gtk/gtk.h>

int main (int argc, char *argv[])
{
 GtkWidget *window;

        /* --- Initialize gtk, handle command-line parameters --- */
        gtk_init (&argc, &argv);

        /* --- Create a window in gtk - note the window is NOT visible yet --- */
        window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

        /* --- Now, make the window visible --- */
        gtk_widget_show (window);
```

```
    /* --- Main event loop in gtk. We have no event handlers. Oops --- */
    gtk_main ();

    /* --- Exit status code --- */
    return 0;
}
```

虽然上面的例子是一个 GTK + 程序，但它不是一个应用程序；因为它没有任何有用的功能，而且当你试图关闭它时它并不退出。在你能够称它为应用程序以前，应该加入回呼函数。

3.1.4　GTK + 数据类型

在开始增加事件使这个小的应用程序更为完善以前，需要了解有关 GTK + 构件类型的知识。在 GTK + 中，通常由构件派生构件。按钮构件（GtkButton）由容器构件（GtkContainer）派生；而容器构件则由通用构件（GtkWidget）派生；而通用构件则又由 GTK + 对象（GtkObject）派生。所有建立构件的函数返回指向 GtkWidget 的指针。这是通用构件的指针，可能需要转换为特定的构件函数。从构件派生构件的理由是简单的：如果一个构件完成另一个构件的大部分功能，为什么还要花时间和精力去写新的构件？一种更容易的解决方案是新的构件由已有的构件派生，并增加必要的功能。

从构件派生构件似乎应该用 C++ 来写 GTK +，但选择 C 有一定的理由。C 语言是 Linux 系统给开发人员提供的主要语言；C 语言比 C++ 语言更容易移植；而且它早已标准化。当开始开发 GTK + 时，C++ 标准化的条件还未成熟。发展 C++ 走过了漫长的道路，C++ 实现标准化的时间还不长。不是所有 C++ 编译软件都能对同一组代码进行编译；而且在跨平台的使用中仍然出现一些问题（现在已有一些在 C++ 库上建立 GTK + 的项目在进行，详细情况见 GTK + 的 Web 页）。

GTK + 的 button-creation（建立按钮）函数返回 GtkWidget 指针，而不是 GtkButton 函数。这种条件使得通用函数（如 gtk _ widget _ show）可以对所有的构件进行操作。从 GTK + 的 button-creation 函数返回的 GtkWidget 指针可以指定为不同的结果。如果它是一个按钮，可以（使用 GTK _ BUTTON 宏）分配 GtkButton 作为专用的按钮函数。如果它是一个容器，可以（使用 GTK _ CONTAINER 宏）分配 GtkContainer 作为容器函数。

虽然可以对按钮构件传递一个专用的按钮函数，但是如果你作为构件来传递，编译软件会给出错误信息。正确的 GTK + 编程要求：需要在调用构件函数以前对构件分配正确的类型。每一种构件有一个转换宏将 GtkWidget 分配为任何其他的 GTK + 构件类型。如果 GtkWidget 能够分配为按钮（这表示 GtkWidget 是由建立按钮函数创建的，或者是由按钮派生的构件），可以使用 GTK _ BUTTON 宏将 GtkWidget 分配为 GtkButton。如果不是这样，虽然编译软件不给出错误信息，在程序运行时，GTK + 库将给出错误信息。GTK + 函数检查传递给它的构件类型。所以，如果传递按钮给容器，函数将正常工作，因为按钮是容器。如果传递容器给按钮，函数将失败，因为容器不是按钮。在控制台上显示这些错误信息，这对建立没有错误的应用很有帮助。

3.1.5 信号和回呼函数

由于程序必须能够对用户作出响应,在基于 GUI 的程序设计中信号是必要的。移动鼠标,按下按钮,敲入正文或者关闭窗口,将给应用软件的回呼函数提供信号。信号可能需要由应用软件来加以处理。字处理软件可能有使字体变黑的按钮。如果用户按下了按钮,就需要调用使字体变黑的程序。与此类似,如果用户关闭了主窗口,在实际关闭窗口以前需要进行某些处理(如保存文件,清除等)。

在 GTK + 中经常产生各种信号,多数情况下信号被忽略。以按钮构件为例,应用软件有专用于按钮的信号。当用户按下鼠标按钮或释放鼠标按钮时,当用户点击鼠标时,或者当鼠标移过按钮或离开按钮时都产生各自的信号。另一类通用构件信号包括标志鼠标移动的信号。

对按钮构件来说,实际上唯有点击鼠标的信号值得监听。这是用户点击鼠标按钮时发出的信号。在多数情况下,应用程序忽略按钮的其他事件。每当用户移动鼠标经过按钮,或释放鼠标按钮时发出通知是没有意义的。其他构件类似于按钮构件,对应用程序开发人员来说只有少量信号是有用的,多数信号则可以被忽略。

当需要对信号进行处理时,需要用 GTK + 登记回呼函数,并将它和构件联系在一起。构件可以登记回呼函数,回呼函数可以与多个构件联系在一起。图 3.2 表示应用程序、GTK + 和回呼函数函数之间的关系。

图 3.2 表示如何处理信号(图中 Event Handler 表示事件处理程序,"click"表示一种信号)。按钮 5 定义了回呼函数,当信号出现时它将被调用。按钮 8 定义了两个不同的回呼函数监听两个不同的信号。允许构件按需要响应多个信号。好的做法是将响应信号的数量保持在最低限度。当在构件中出现信号时,就执行构件中的回呼函数。

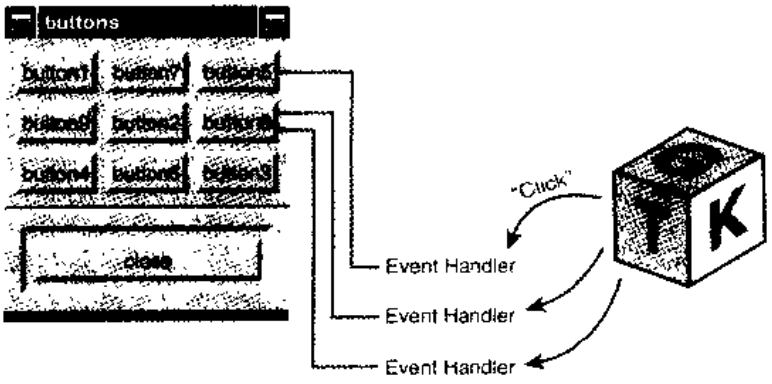


图 3.2 GTK + 的信号处理

3.1.6 增加信号处理程序

GTK + 的两个基本信号是 delete _ event 和 destroy 信号。当将要关闭窗口时,发出 delete _ event 信号。当关闭窗口时,发出 destroy 信号。对于 delete _ event 信号,顶层窗口应该设有相应的回呼函数。因为 delete _ event 信号表示用户需要关闭应用软件。对 delete _ event 信号

增加回呼函数包括两个步骤。

第一步是写回呼函数，在本例中称它为 destroyapp。destroyapp 函数在屏幕上显示消息，并调用 gtk _ main _ quit 函数退出事件循环。必须调用 gtk _ main _ quit 函数结束应用软件。如果在关闭顶层窗口前不退出事件循环，实际上程序将继续运行和处理事件。这就是为什么需要调用 gtk _ main _ quit 函数的原因。delete _ event 的回呼函数应该返回布尔值，用以表示是否允许关闭窗口。返回 TRUE 值保持窗口打开；返回 FALSE 值表示窗口可以关闭。如果可以关闭窗口，接着发出 destroy 信号告诉应用程序就要关闭窗口。

第二步是使用 gtk _ signal _ connect 函数向 GTK + 登记事件处理函数。gtk _ signal _ connect 函数有 4 个参数。用这些参数告诉 GTK + ：回呼函数用于哪一个构件，用回呼函数处理什么信号；当出现信号时用哪一个函数作为回呼函数，以及调用信号处理程序需要传递的附加参数。

```
/* --- Call destroyapp when the window gets "delete_event" signal--- */
gtk_signal_connect (GTK_OBJECT (window),"delete_event",
        GTK_SIGNAL_FUNC (destroyapp), NULL);
```

注意：在调用 gtk _ signal _ connect 函数时，由 GTK _ OBJECT 分配窗口。gtk _ signal _ connect 要求给它传递 GtkObject。在构件的层次结构中，由于 GtkWindow 是由 GtkObject 派生的，可以分配 GtkWindow 给 GtkObject。

对程序解释如下：如果在窗口构件上出现了 delete _ event 信号，将第 4 个参数设置为 NULL 调用 destroyapp 函数。

增加了事件处理程序后，完整的程序如下：

```
#incluce <gtk/gtk.h>

gint destroyapp (GtkWidget *widget, gpointer gdata)
{
    g_print ("quitting---\n");
    gtk_main_quit ();

    /* --- Ok to be closed. TRUE would prevent window--- */
    /* --- from closing--- */
    return (FALSE);
}

int main (int argc, char *argv[])
{
    GtkWidget *window;

    /* --- Initialize gtk, handle command-line parameters--- */
    gtk_init (&argc, &argv);
```

```
    /* --- Create a window in gtk - note the window is NOT visible yet --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- Call destroyapp when the window gets a "delete_event" signal */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (destroyapp), NULL);
    /* --- Now, make the window visible --- */
    gtk_widget_show (window);

    /* --- This is the event loop in gtk --- */
    /* --- Do not return until _gtk_main_quit_ is called --- */
    gtk_main ();

    /* --- Exit status code --- */
    return 0;
}
```

注意,当关闭窗口时,将在启动应用程序的控制台上显示 Quitting…消息。这是由回呼函数显示的。在主程序中,调用 gtk_main 函数后进入事件循环处理事件,直到你试图关闭窗口为止。当你试图关闭窗口时,destroyapp 函数被调用,由它通知应用程序出现了 delete_event 信号。通过 gtk_signal_connect 函数向 GTK + 登记 destroyapp 函数。destroyapp 函数知道关闭主窗口就是结束应用程序。它调用 gtk_main_quit 函数结束在应用程序中启动的 gtk_main 循环。现在程序已成为简单而功能完善的 GTK + 应用程序。在前一个程序中,虽然同样出现 delete_event 信号,但是应用程序却没有事件处理程序来处理它。现在,这一信号由回呼函数进行处理,应用程序也就正常退出。

3.2 增加构件

没有菜单或控制的空白窗口也可以称为一项应用,但是它决不是有用的应用。你可以在前面建立的空白窗口中添加功能来开始写一项应用。在这种情况下,可以在窗口中增加构件。由于已经建立的顶层窗口是一个容器,可以在它里面放置构件。使用 gtk_button_new_with_label 函数建立按钮,并用 gtk_container_add 函数将按钮放置到窗口中。注意:顶层窗口和按钮都必须使用 gtk_widget_show 函数使它们成为可见。但是还需要通过附加的步骤将按钮加到容器中。在将按钮加入容器以前,即使它已经被建立和成为可见,它在屏幕上仍然是看不见的。构件必须在顶层窗口(无上层窗口或父窗口的窗口)中,或者它们必须在可见的父窗口中成为可见。函数 gtk_container_add 建立顶层窗口和按钮之间的父/子关系,并将构件放置在窗口中。还可以给按钮增加一个处理点击事件的事件处理程序,使它在每次按按钮时调用 button_was_clicked 函数,并在屏幕上显示一条消息。图 3.3 显示一个按钮,每次按下按钮时将显示一条消息。



图 3.3　GTK+ 按钮的简单事件。

```
#include <gtk/gtk.h>

gint CloseAppWindow (GtkWidget *widget, gpointer gdata)
{
    g_print ("Quitting---\n");
    gtk_main_quit ();

    /* --- Continue closing --- */
    return (FALSE);
}

/*
 * button_was_clicked
 *
 * event handler called when the button is clicked.
 */
void button_was_clicked (GtkWidget *widget, gpointer gdata)
{
    g_print ("Button was clicked. \n");
}

    int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *button;

    /* --- Initialize gtk, handle command-line parameters --- */
    gtk_init (&argc, &argv);

    /* --- Create a window in gtk - note the window is NOT visible yet --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

```
    /* --- Clean up code when the application exits --- */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (destroy), NULL);

    /* --- Put some breathing room around the objects in the container --- */
    gtk_container_border_width (GTK_CONTAINER (window), 15);

    /* --- Create a button --- */
    button = gtk_button_new_with_label ("click me");

    /* --- Give the button an event handler - it'll call the function --- */
    /* --- button_was_clicked when the button is clicked --- */
    gtk_signal_connect (GTK_OBJECT (button),
                        "clicked",
                        GTK_SIGNAL_FUNC (button_was_clicked),
                        NULL);

    /* --- The main window contains the button --- */
    gtk_container_add (GTK_CONTAINER (window), button);

    /* --- Make the button visible --- */
    gtk_widget_show (button);

    /* --- Now, make the window visible --- */
    gtk_widget_show (window);

    /* --- This is the event loop in gtk. Do not return until "Quit" --- */
    gtk_main ();

    /* --- Exit status code --- */
    return 0;
}
```

用几个步骤将构件加到窗口中。

第 1 步建立构件,并取得 GtkWidget 指针。

第 2 步对构件设置回呼函数。一些构件不能有回呼函数(如标号);一些构件不需要回呼函数;但是,许多构件用回呼函数与用户进行对话。

第 3 步设置构件的特性。例如,对按钮设置标号,或使标号左对齐等。有时可以用建立构件的函数设置特性。例如,可以用 gtk_button_new 函数建立按钮,也可以用 gtk_button_new_with_label 函数建立带标号的按钮。如果要建立带标号的按钮,使用 gtk_button_new

_ with _ label 函数更为方便。因为它不需要分别建立按钮和标号，再将标号加到按钮中去。它一步就完成这些功能。

第 4 步将构件加到容器中。除非构件是顶层窗口，所有其他构件都必须将它们加到可视容器内才能被显示。

最后一步是使构件成为可见。使构件成为可见不一定要放在最后。因为它改变了窗口的性质，通常将它放在最后。当它成为可见时，将出现恼人的闪烁；特别是在不能快速显示的慢的机器上，情况更是如此。

3.2.1　多事件处理程序

在许多情况下，构件需要用多事件处理程序来处理不同的事件。多事件处理程序只是已有的单事件处理程序的扩充。本节将扩充前面提到的按钮的例子，并增加处理鼠标移到按钮上和离开按钮的事件。取决于鼠标移到按钮或离开按钮，事件处理程序将在控制台上显示不同的正文。需要建立处理鼠标进入和离开按钮事件的程序。

```
/*
 * button_entered
 *
 * Called when the mouse moves over the widget after not having
 * been on the widget.
 */
void button_enter (GtkWidget *widget, gpointer gdata)
{
      g_print ("welcome to fantasy button \n");
}

   /*
 * button_leave
 *
 * Called when the mouse moves off the widget.
 */
void button_leave (GtkWidget *widget, gpointer gdata)
{
      g_print ("hey! Where are you going? \n");
}
```

在写完回呼函数以后，需要向 GTK + 登记。这样，当发生信号时才能调用回呼函数。可以用 gtk_signal_connect 函数登记回呼函数，并将它和构件的信号联系起来。进入(enter)事件表示鼠标移动到按钮上，离开(leave)事件表示鼠标离开按钮。将这两个事件指定给回呼函数，就能建立响应 3 种不同事件的按钮。

```
/* --- Give the button an event handler - it'll call the function ---*/
/* --- button_enter when the mouse moves over the button --- */
```

```
gtk_signal_connect (GTK_OBJECT (button),
                    "enter",
                    GTK_SIGNAL_FUNC (button_enter),
                    NULL);

/* ---Give the button an event handler - it'll call the function---*/
/* ---button_leave when the button moves off the button--- */
gtk_signal_connect (GTK_OBJECT (button),
                    "leave",
                    GTK_SIGNAL_FUNC (button_leave),
                    NULL);
```

现在,按钮有 3 个回呼函数,每一个分别用来处理来自 GTK + 的不同的信号,并根据具体信号显示不同的消息。当移动鼠标经过按钮和点击按钮时,你可以看到如何发送信号和回呼函数如何对信号作出响应。

3.3 容器

你已经看到顶层窗口如何作为容器,并将构件放置在里面。当你知道许多构件可以作为容器时,也许会感到惊讶! 因为按钮是一个容器,其他构件可以放在按钮构件中。事实上,在使用 gtk_button_new_with_label 函数建立带标号的按钮构件时,在按钮构件中已经放入了标号构件。

下列限制适合于 GTK + 的各种容器:容器只能有一个子构件。为什么是这样,如果你在前面的例子中增加第二个按钮,只有第一个按钮是可见的。绕过这一限制的办法是使用组合构件。

3.3.1 组合框

使用组合构件(packing widget)可以将多个构件放在一个容器中。组合构件也称为组合框(packing box)。不像按钮构件,组合构件在屏幕上是看不见的。事实上,它们是同时跟踪若干子构件的不可见容器。纵向组合框(vertical packing box)在垂直方向堆积构件,横向组合框(horizontal packing box)沿水平方向堆积构件。使用 gtk_vbox_new 函数建立纵向组合框,使用 gtk_hbox_new 函数建立横向组合框。在建立组合框以后,可以使用 gtk_box_pack_start 函数或 gtk_box_pack_end 函数将构件放到组合框中。函数 gtk_box_pack_start 将构件放在顶部(对纵向组合框)或左边(对横向组合框)。函数 gtk_box_pack_end 将构件放在底部(对纵向组合框)或右边(对横向组合框)。为了显示构件,必须将构件放入组合框中,并将组合框放在容器内。这等效于使容器具有容纳多个构件的能力。虽然组合框没有任何可视的部分,但是必须使容器和组合框成为可视。可视性影响加到容器内的构件。图 3.4 显示放在纵向和横向组合框内的按钮。

建立组合框的函数取两个参数,用来确定组合框在屏幕上的外观。

```
/* ----Create a horizontal packing box--- */
hbox = gtk_hbox_new (bHomogeneous, nSpacing);


/* ---Create a vertical packing box--- */
vbox = gtk_vbox_new (bHomogeneous, nSpacing);
```

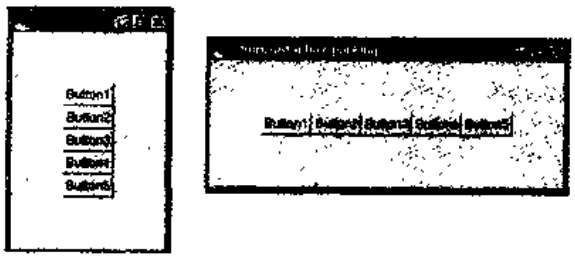


图 3.4　横向和横向组合框的例子

第一个参数称为 homogeneous。用这一参数来确定所有增加的构件是否在屏幕上占同样的空间。例如，在一个横向组合框中建立 5 个按钮，分别带有有标号：fred、joe、sue、karen 和 Bartholomew the Great。非 homogeneous 组合框建立 5 个不同大小的按钮，正好在按钮上容纳各自的标号的正文。homogeneous 组合格式计算最大的构件的大小，然后按最大的构件的大小对所有按钮分配同样大小的空间，不管按钮是否充分利用了空间。第二个参数称为 spacing。用它来确定插入组合框的构件之间的空间。将 spacing 参数设置为 0 表示在插入的构件之间不留空间。

函数 gtk_box_pack_start 用 3 个参数来确定如何将构件放在组合框内。

```
gtk_box_pack_start (box, widget, expand, fill, padding);
```

expand 参数表示在所有构件加入组合框以后，构件周围是否还有可扩充的空间。如果用 homogeneous 格式建立组合框，则忽略这一参数。因为 homogeneous 参数表示构件在组合框内使用同样大小的空间。

fill 参数表示构件是否需要充分利用构件周围空间。将此参数设置为 TRUE 允许构件稍稍扩大一点，以充分利用组合框分配给它的空间。将此参数设置为 FALSE 强制构件只使用它需要的空间。多余的空间围绕构件的周围分布。

padding 参数表示在构件周围要保留多少个填充的像元。多数情况下将它的值设置为 0。

下面的例子表示如何建立包含 5 个按钮的纵向组合框的窗口。为了简化程序，将建立和增加按钮的代码合并为一个称为 PackNewButton 的函数。

```
/*
 * packbox.c
 *
 * Program illustrating the difference between vertically packed boxes
 * and horizontally packed boxes by displaying two windows each with
 * their own packing.
```

```
 */

#include <gtk/gtk.h>

 * PackNewButton
 *
 * Creates a button, packs it into the box provided and makes the
 * button visible.
 *
 * box  packing box to add the button to.
 * szLabel - label the button will be created with.
 * returns - button widget that was created.
 */
GtkWidget * PackNewButton (GtkWidget * box, char * szLabel)
{

    GtkWidget *button;

    /* --- Create a new button --- */
    button = gtk_button_new_with_label (szLabel);

    /* --- Pack the button into the box --- */
    gtk_box_pack_start (GTK_BOX (box), button, FALSE, FALSE, 0);

    /* --- Show the button --- */
    gtk_widget_show (button);
        return (button);
}

/*
 * delete_event
 *
 * Handle closing of window properly by ending gtk
 */
gint Delete (GtkWidget *widget, gpointer *data)
{
    gtk_main_quit ();

    /* --- Continue closing --- */
    return (FALSE);
}
```

```
int main (int argc, char *argv[])
{
    GList *cbitems = NULL;
    GtkWidget *window;
    GtkWidget *button;
    GtkWidget *box;

    /* --- GTK initialization --- */
    gtk_init (&argc, &argv);
    /* ------------------------------- */
    /* --- Create the vertically packed box. --- */
    /* ------------------------------- */

    /* --- Create the top-level window --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- Give window a title --- */
    gtk_window_set_title (GTK_WINDOW (window),
                          "Vertical box packing");
    /* --- Want to know when the window closes --- */
    /* --- to the main window --- */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (Delete), NULL);

    /* --- Give the window a border --- */
    gtk_container_border_width (GTK_CONTAINER (window), 50);

    /* --- We create a vertical box to pack the buttons --- */
    box = gtk_vbox_new (FALSE, 0);

    /* --- Create the buttons --- */
    button = PackNewButton (box, "Button1");
    button = PackNewButton (box, "Button2");
    button = PackNewButton (box, "Button3");
    button = PackNewButton (box, "Button4");
    button = PackNewButton (box, "Button5");

    /* --- Make the main window visible --- */
    gtk_container_add (GTK_CONTAINER (window), box);
    gtk_widget_show (box);
    gtk_widget_show (window);
```

```
    /* ------------------------------*/
    /* --- Create a horizontally packed box. --- */
    /* ------------------------------ */

    /* --- Create the top-level window --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- Give the window a title --- */
    gtk_window_set_title (GTK_WINDOW (window), "Horizontal box packing");

    /* --- Give the window a border --- */
    gtk_container_border_width (GTK_CONTAINER (window), 50);

    /* --- We create a horizontal box to pack the buttons --- */
    box = gtk_hbox_new (FALSE, 0);

    /* --- Create the buttons --- */
    button = PackNewButton (box, "Button1");
    button = PackNewButton (box, "Button2");
    button = PackNewButton (box, "Button3");
    button = PackNewButton (box, "Button4");
    button = PackNewButton (box, "Button5");

    /* --- Make the main window visible --- */
    gtk_container_add (GTK_CONTAINER (window), box);
    gtk_widget_show (box);
    gtk_widget_show (window);
    /* --- Start the event loop --- */
    gtk_main ();
    exit (0);
}
```

可以将组合框放入其他组合框内以建立充满构件的屏幕,而不是只有一组纵向或横向组合构件的屏幕。通常将一行构件放在横向组合框内,再将横向组合框和横向组合框一起放入纵向组合框内,这一过程建立若干行包含构件的横向组合框,可以用来组织整个屏幕。

3.3.2 组合表

组合表(packing table)允许将多个构件放入一个窗口内,这一点类似于组合框。但是,组合框在放置构件时只允许纵向或横向的二维控制。组合表则类似于 HTML 表:允许用行和列控制构件的放置,并且对象可以取多行或多列。建立组合表的函数如下:

```
table = gtk_table_new (rows, columns, homogeneous);
```

前两个参数表示建表所用的行数和列数。第 3 个参数 homogeneous 的效果和在组合框中使用的的效果相同。如果在建立组合表时它被设置为 TRUE,表内的表框用最大构件的外框。如果它被设置为 FALSE,则将每一列的宽度设置为本列中最大构件的宽度,每一行的高度设置为行中最大构件的高度。

行的排列从 0…rows - 1,列的排列从 0…columns - 1。

当增加构件时,每个构件提供开始列、结束列、开始行和结束行。如果构件将被放在表的左上角,列的范围从 0 到 1,行的范围从 0 到 1。Xpadding 和 ypadding 表示围绕构件填充的像元数。

两个用来将构件加到表中的函数如下:

```
gtk_table_attach(table,
                 child,
                 left_attach,
                 right_attach,
                 top_attach,
                 bottom_attach,
                 xoptions,
                 yoptions,
                 xpadding,
                 ypadding);

gtk_table_attach_defaults(table,
                          widget,
                          left_attach,
                          right_attach,
                          top_attach,
                          bottom_attach);
```



图 3.5　组合表的选项

两个函数函数都将构件加到表中。gtk_table_attach_defaults 函数取比较少的参数,并对 gtk_table_attach 函数使用的 options(选项)和 padding(填充)参数用缺省值来代替。对 gtk_table_attach_defaults 函数来说,xpadding 和 ypadding 的缺省值为 0;而 xoptions 和 yop-

tions 的缺省值为(GTK _ FILL | GTK _ EXPAND)。下面的程序表示一个 4 * 4 表用不同的选项值产生不同的结果(见图 3.5)。每个窗口放一个按钮,使用选项的 4 种可能设置。当重新改变窗口大小时,将按照每个窗口放入构件时使用的选项值重新显示构件。

选项 xoptions 和 yoptions 可以用 GTK _ FILL、GTK _ SHRING 和 GTK _ EXPAND,或者用 3 者的任意组合。GTK _ FILL 选项表示构件充分利用分配给它的空间进行扩展。GTK _ SHRING 选项允许构件缩小到比原来分配的空间还小的空间。GTK _ EXPAND 选项使表扩展填满它插入的窗口的所有空间。

在程序中,左上角的按钮在表中占 1 平方(0-1,0-1);右上角和左下角的按钮各占 2 平方;而右下角的按钮占 4 平方。在改变窗口大小时,可以看到这些属性所产生的影响。所有建立窗口的代码都放在 CreateTable 函数中。

```
/*
 * packtable
 *
 * Illustration of how to pack tables.
 *
 */

#include <gtk/gtk.h>

/*
 * Destroy
 *
 * Called when the main window is being closed. Gives us the
 * opportunity to exit from the call to gtk_main ()
 */
gint Delete (GtkWidget *widget, gpointer data)
{
    gtk_main_quit ();

    /* --- Continue closing --- */
    return (FALSE);
}

/*
 * CreateTable
 *
 * Creates a top-level window with several options. Since
 * this is going to be done many times, it's best just made
 * into a function.
 */
```

```
CreateTable (char * szTitle, gint xoptions, gint yoptions)
{
    GtkWidget * window;
    GtkWidget * button;
    GtkWidget * table;

    /* --- Create the top-level window --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- Give the window a title --- */
    gtk_window_set_title (GTK_WINDOW (window), szTitle);

    /* --- You should always remember to connect the destroy event --- */
    /* --- to the main window. Note: if any top-level window is --- */
    /* --- closed, we'll shut the whole program down --- */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (Delete), NULL);

    /* --- Give the window a border --- */
    gtk_container_border_width (GTK_CONTAINER (window), 10);

    /* --- Create a 4x4 table --- */
    table = gtk_table_new (4, 4, TRUE);

    /* --- Add a 1x1 button at 0, 0 --- */
    button = gtk_button_new_with_label ("Button at 0,0");
    gtk_table_attach (GTK_TABLE (table), button, 0, 1, 0, 1,
                      xoptions, yoptions, 0, 0);
    gtk_widget_show (button);

    /* --- Add a 2x2 button at 2-4, 2-4 --- */
    button = gtk_button_new_with_label "Big square button");
    gtk_table_attach (GTK_TABLE (table), button, 2, 4, 2, 4,
                      xoptions, yoptions, 0, 0);
    gtk_widget_show (button);

    /* --- Add a 2x1 short wide button --- */
    button = gtk_button_new_with_label ("Short fat button");
    gtk_table_attach (GTK_TABLE (table), button, 0, 2, 3, 4,
                      xoptions, yoptions, 0, 0);
    gtk_widget_show (button);
```

```
    /* --- Add a 1x2 tall skinny button --- */
    button = gtk_button_new_with_label "Tall skinny button");
    gtk_table_attach (GTK_TABLE (table), button, 3, 4, 0, 2,
                      xoptions, yoptiors, 0, 0):

    /* --- Make the button visible --- */
    gtk_widget_show (button);

    /* --- Add the table to the window --- */
    gtk_container_add (GTK_CONTAINER (window), table);

    /* --- Make the table visible --- */
    gtk_widget_show (table);

    /* --- Make the window visible --- */
    gtk_widget_show (window);
}

/*
 * --- Main.
 *
 * Program begins here.
 *
 * Rather than duplicate code, the CreateTable function is
 * called. It does the work for us - all we do is pass in
 * how we want the window to appear.
 */
int main (int argc, char *argv[])
{

    /* --- GTK initialization --- */
    gtk_init (&argc, &argv);

    /* --- No flags set --- */
    CreateTable ("Nothing", 0, 0);

    /* --- Start showing windows with flags --- */
    CreateTable ("Hill", GTK_FILL, GTK_FILL);

    CreateTable ("Shrink", GTK_SHRINK, GTK_SHRINK);

    CreateTable ("Expand",
```

```
                GTK_EXPAND,
                GTK_EXPAND);

    CreateTable ("Expand | Shrink",
                GTK_EXPAND | GTK_SHRINK,
                GTK_EXPAND | GTK_SHRINK);

    CreateTable ("Fill | Expand",
                 GTK_EXPAND | GTK_FILL,
                 GTK_EXPAND | GTK_FILL);

    CreateTable ("Shrink | Fill",
                GTK_SHRINK | GTK_FILL,
                GTK_SHRINK | GTK_FILL);

    CreateTable ("Shrink | Fill | Expand",
                GTK_EXPAND | GTK_SHRINK | GTK_FILL,
                GTK_EXPAND | GTK_SHRINK | GTK_FILL);

    /* --- Start the gtk loop --- */

    gtk_main ();

    return (0);

}
```

3.4　小结

本章讨论了 GTK + 程序设计基础，包括编写演示 GTK + 编程结构的简单应用。并讨论了基本的 GTK + 事件和构件，以及一些 GTK + 数据类型。这些信息为讨论下一章（基本构件）打下了必要的基础。

第4章 基本构件

建立应用软件要求使用多种不同构件。最常用的构件是按钮构件、录入字段构件、列表框构件和复合框构件。这些构件在应用中占主要的比例。了解这些基本构件有助于进一步掌握更复杂的构件。事实上,大多数构件非常类似。许多构件只是其他具有特殊性质的构件的容器;它们具有可以用来处理这些构件的共同特性和功能。

4.1 常用的构件函数

所有建立按钮的函数返回指向 GtkWidget 的指针。GtkWidget 指针能够调用对构件进行操作的通用函数。函数 gtk _ widget _ show 是通用函数的例子。这一函数使构件成为可视;或者至少对构件设置可视属性。当将构件置于可视容器内时,可以在屏幕上看到它。许多其他函数也对所有构件进行操作。其中一些函数(如 gtk _ widget _ show)比其他函数用得更为频繁。

4.1.1 转换构件的类型

由于建立的构件是通用的构件,需要将它转换为正确的类型以适合调用更为专用的函数。建立按钮构件函数返回 GtkWidget 指针,但是专用的按钮子程序要求 GtkButton 指针。所以,在调用专用的按钮函数以前,需要使用 GTK _ BUTTON 宏将通用的 GtkWidget 指针转换为 GtkButton 指针。坚持传送的类型符合函数的具体的要求,将使开发人员写出更好的程序。

4.1.1.1 拙劣的转换

只有写得很差的程序才会将正文构件传送给要求按钮构件的函数。为了强制进行必要的转换,将报告在转换构件的过程中发生的任何错误。例如,下面的程序建立按钮构件,并将它转换为正文构件。

```
/* --- Create a button widget --- */
button = gtk_button_new_with_label ("Button");

/* --- Try and convert the button to a text widget --- */
text = GTK_TEXT (button);
```

如果将这段代码放在程序中,由于转换检查放在运行时进行,所以在编译时没有任何反应。但是在运行程序时将出现如下的错误消息:

```
** WARNING **: invalid cast from "GtkButton" to "GtkText"
```

在正常情况下不应该出现这样的消息。如果出现这样的消息,通常是由开发人员的错误

造成的。

4.1.1.2 好的转换

通常构件是由其他构件派生。按钮构件由容器构件(GtkContainer)派生,而容器又由通用构件(GtkWidget)派生。由于这种关系,按钮构件(GtkButton)可以分配给容器构件(GtkContainer)以利用容器函数。你已经用过 gtk_container_add 函数将构件加到容器中去。还可以将这一函数用于按钮构件,将标号加到按钮中去。当然,为了利用 gtk_container_add 函数,必须将按钮分配给 GtkContainer;这是可能的,因为 GtkButton 也是 GtkContainer。可以在上层构件的范围内将构件转换为任何构件类型。

```
/* --- Create a button without a label --- */
button = gtk_button_new ( )

/* --- Creatw a label --- */
label = gtk_label_new ("Blah");

/* --- Add the label to the button --- */
gtk_container_add (GTK_CONTAINER (button), label);
```

每个构件有它自己的一组信号;开发人员可以对信号建立回呼函数。与每个构件联系在一起的信号数量取决于构件的复杂性;一般说来在应用中只有很少的信号需要处理。按钮构件就是很好的例子。虽然按钮构件有许多信号和它联系在一起,通常只有一个信号值得对它建立事件处理程序,这就是"点击"事件。这不是说从不使用其他事件。更主要的含义是经常不使用它们。事件能够被继承,当你查看从按钮构件继承的构件时会发现这一特征。

4.2 按钮

按钮(push button,GtkButton)是最简单的构件,因为它们的唯一功能是允许用户点击它们。在对话过程中,它们通常显示如 OK 或 Cancel 一类的正文。对按钮来说,最重要的事件是"点击"事件,它表示用户用鼠标点击了按钮。"点击"事件是一次按下按钮和释放按钮的组合。通常点击按钮导致某些操作的完成,如保存文件或结束对话等。图 4.1 是按钮的例子。

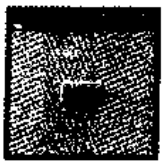


图 4.1 按钮

按钮由容器(GtkConyainer)派生而来,所以它们共享容器的许多特性。一个重要的特性是容纳其他构件的能力。在 GtkButton 内的正文实际是放在按钮构件内的标号构件。可以用

一排按钮来建立工具条(toolgar),用每个按钮的内置影像向用户说明它的功能。初看之下它是非常乏味的构件,现在看来它非常灵活。事实上,许多其他构件从 GtkButton 构件派生出来。

建立按钮可以带正文或不带正文。函数 gtk _ button _ new _ with _ label 建立带字符串说明的按钮。函数 gtk _ button _ new 建立不带子构件的按钮。当你需要建立带影像的按钮时,这一函数是有用的。由于你不需要标号,你可以用 gtk _ button _ new 函数建立按钮,并将影像加到按钮中去。函数返回指向 GtkWidgetd 指针,可以使用 GTK _ BUTTON 宏将它转换为 GtkButton。

按钮构件能够发送若干种信号。但是,除了"点击"信号以外,开发人员忽略大部分信号。按钮信号有:

信号	操作
pressed	按下按钮
released	释放按钮
clicked	点击按钮,这是"按下"和"释放"操作的组合
enter	鼠标移到按钮上
leave	鼠标离开按钮

事件可能是用户的操作引起的,也可以用下面的函数模拟事件的发生。但不经常使用模拟信号发生的函数。

信号	产生信号的函数
pressed	gtk_button_pressed (button)
released	gtk_button_released (button)
clicked	gtk_button_clicked (button)
enter	gtk_button_enter (button)
leave	gtk_button_leave (button)

下面的短程序建立带按钮的窗口。当产生信号时,在控制台上显示每一种按钮信号。

```
/*
 * button.c
 *
 * Example showing button behavior
 */

# include <gtk/gtk.h>

/*
 * Destroy
```

```
 *
 * The app is closing down. Need to shut down gtk.
 */
gint CloseAppWindow (GtkWidget *widget, gpointer *data)
{
    gtk_main_quit ();

    /* --- Ok to close --- */
    return (FALSE);
}

/*
 * button_event
 *
 * Some event happened and the name is passed in the
 * data field.
 */
void button_event (GtkWidget *widget, gpointer *data)
{
    g_print ("Button event: %s\n", data);
}

int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *button;

    /* --- GTK initialization --- */
    gtk_init (&argc, &argv);

    /* --- Create the top-level window --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    gtk_window_set_title (GTK_WINDOW (window), "push Button");

    /* You should always remember to connect the delete_event
      * to the main window
      */
    gtk_signal_connect (GTK_OBJECT (window), "Delete_event",
                        GTK_SIGNAL_FUNC (CloseAppWindow), NULL);

    /* --- Give the window a border --- */
```

```
    gtk_container_border_width (GTK_CONTAINER (window), 50);

    /* ------------------ */
    /* --- Create a button --- */
    /* ---------------------*/

    /* --- Create a new button --- */
    button = gtk_button_new_with_label "Button");

    /* --- Show the button (it's not visible yet, though) --- */
    gtk_widget_show (button);

    /* ------------------------ */
    /* --- Register event handlers --- */
    /* ------------------------ */
    gtk_signal_connect (GTK_OBJECT (button), "pressed",
                        GTK_SIGNAL_FUNC (button_event), "pressed");
    gtk_signal_connect (GTK_OBJECT (button), "released",
                        GTK_SIGNAL_FUNC (button_event), "released");
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (button_event), "clicked");
    gtk_signal_connect (GTK_OBJECT (button), "enter",
                        GTK_SIGNAL_FUNC (button_event), "enter");
    gtk_signal_connect (GTK_OBJECT (button), "leave",
                        GTK_SIGNAL_FUNC (button_event), "leave");

    /* --- Add the button to the window --- */
    gtk_container_add (GTK_CONTAINER (window), button);

    /*
     * --- Make the main window visible, now the button
     *     will be made visible.
     */
    gtk_widget_show (window);

    gtk_main ();
    return (0);
}
```

4.3 触发按钮

触发按钮(toggle button,GtkToggleButton)是由 GtkButton 派生而来。它的外观类似 Gtk-

Button,但运作方式稍有不同。触发按钮与一种状态结合在一起。它可以按下和弹回,触发按钮的外观反映它的状态。在开始,触发按钮看上去和普通按钮一样。如果按下按钮,它就停留在下面。触发按钮需要再按一次才能弹回来。图 4.2 表示触发按钮的按下和弹回两种状态。



图 4.2　触发按钮

可以使用 gtk _ toggle _ button _ new _ with _ labal 函数建立带标号的触发按钮,或者使用 gtk _ toggle _ button _ new 函数建立不带标号的触发按钮。触发按钮由普通按钮派生而来。如果你用 GTK _ BUTTON 宏分配触发按钮,所有可以用在普通按钮上的事件和函数都可以用在触发按钮上。

由 GtkButton 派生的触发按钮增加了一种新的信号。当按钮的状态改变时它发送"触发"信号。可以用 gtk _ toggle _ button _ set _ state 函数设置触发按钮的状态,它的参数是需要的触发按钮状态。函数 gtk _ toggle _ button _ set _ state 改变按钮的状态。改变按钮状态的函数同样能将信号发送给应用。例如,如果函数 gtk _ toggle _ button _ set _ state 将按钮的状态设置为 TRUE,而按钮的当前状态是 FALSE,则将"点击"信号和"触发"信号同时发送给应用。

```
/*
 * togglebutton.c
 *
 * Example showing toggle button behavior
 */

#include <gtk/gtk.h>

/*
 * Destroy
 *
 * main window closing, end gtk
 */
gint CloseAppWindow (GtkWidget *widget, gpointer *data)

    gtk_main_quit ();

    /* ---Ok to close--- */
    return (FALSE);
}
```

```
    /* --- Event happened --- */

void ButtonEvent (GtkWidget *widget, button_event gpointer *data)
{
    g_print ("Button event: %s\n", data);
}

int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *button;
    GtkWidget *vbox;

    /* --- GTK initialization --- */
    gtk_init (&argc, &argv);

    /* --- Create the top-level window --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    gtk_window_set_title (GTK_WINDOW (window), "Toggle Button");

    /* --- You should always remember to connect the destroy event --- */
    /*       to the main window --- */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (CloseAppWindow), NULL);

    /* --- Give the window a border --- */
    gtk_container_border_width (GTK_CONTAINER (window), 50);

    /* We create a vertical box (vbox) to pack the horizontal boxes into.
    vbox = gtk_vbox_new (FALSE, 0);

    /* ------------------ */
    /* --- Create a button --- */
    /* ------------------- */

    /* --- Create a new button --- */
    button = gtk_toggle_button_new_with_label "top Toggle Button");

    /* --- Pack the button into the vertical box (vbox box1) --- */
    gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
```

```
/* --- Show the button --- */
gtk_widget_show (button);

gtk_signal_connect (GTK_OBJECT (button), "toggled",
                    GTK_SIGNAL_FUNC (ButtonEvent), "top toggled");
gtk_signal_connect (GTK_OBJECT (button), "pressed",
                    GTK_SIGNAL_FUNC (ButtonEvent), "top pressed");
gtk_signal_connect (GTK_OBJECT (button), "released",
                    GTK_SIGNAL_FUNC (ButtonEvent), "top released");
gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (ButtonEvent), "top clicked");
gtk_signal_connect (GTK_OBJECT (button), "enter",
                    GTK_SIGNAL_FUNC (ButtonEvent), "top enter");
gtk_signal_connect (GTK_OBJECT (button), "eave",
                    GTK_SIGNAL_FUNC (ButtonEvent), "top leave");

/* ---------------------- */
/* --- Create another button --- */
/* ---------------------- */

/* --- Create a new button --- */
button = gtk_toggle_button_new_with_label "Bottom Toggle Button");

/* --- Pack the button into the vertical box (vbox box1) --- */
gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);

/* --- Show the button --- */
gtk_widget_show (button);

gtk_signal_connect (GTK_OBJECT (button), "toggled",
                    GTK_SIGNAL_FUNC (ButtonEvent), "tottom toggled");
gtk_signal_connect (GTK_OBJECT (button), "ressed",
                    GTK_SIGNAL_FUNC (ButtonEvent), "Bottom pressed");
gtk_signal_connect (GTK_OBJECT (button), "released",
                    -GTK_SIGNAL_FUNC (ButtonEvent), "Bottom released");
gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (ButtonEvent), "Bottom clicked");
gtk_signal_connect (GTK_OBJECT (button),"enter",
                    GTK_SIGNAL_FUNC (ButtonEvent), "Bottom enter");
gtk_signal_connect (GTK_OBJECT (button), "leave",
                    GTK_SIGNAL_FUNC (ButtonEvent), "Bottom leave");
```

```
    /*                                                    */
    /*   Make the main window visible                     */
    /* ---------------------------------------------------*/

    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show (vbox);
    gtk_widget_show (window);

    gtk_main ();
    return (0);
}
```

4.4 检查按钮

检查按钮(check button, GtkCheckButton)由触发按钮(GtkToggleButton)派生而来。GtkCheckButton 的表现类似 GtkToggleButton。两者的主要差别是按钮在屏幕上的显示方式不同(见图 4.3)。功能上,两种构件都能够显示正文来说明构件。GtkToggleButton 在触发按钮内显示构件正文,而 GtkCheckButton 则在一个小按钮的右边显示正文。两种按钮完成同样的功能,选择用哪一种按钮纯粹是用户的爱好。

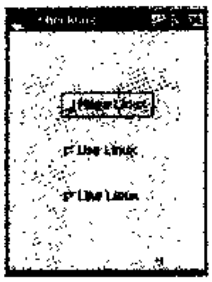


图 4.3 检查按钮

可以用 gtk_check_button_new_with_label 函数建立带标号的 GtkCheckButton,或者用 gtk_check_button_new 函数建立不带标号的 GtkCheckButton。由于 CtkCheckButton 是由 GtkToggleButton 派生而来,所有用于 GtkToggleButton 的函数和事件同样可以用于 GtkCheckButton。

只要替换建立触发按钮的代码,可以将触发按钮的程序转换为检查按钮的例子。旧的建立触发按钮的代码:

```
/* ---Create a new button--- */
button = gtk_toggle_button_new_with_label ("Top Toggle Button");
```

可以用下面的建立检查按钮的代码代替。

```
/* --- Create check button --- */
check = gtk_check_button_new_with_label (szLabel);
```

现在,前面的触发按钮的例子已成为说明检查按钮的例子。

4.5 选择按钮

初看之下,选择按钮(radio button, GtkRadioButton)很像检查按钮。从选择按钮的外观看,选择按钮是从 GtkCheckButton 派生而来。GtkRadioButton 的表现和 GtkCheckButton 差别不大(见图 4.4)。对 GtkRadioButton 来说,任何时候在一组按钮中只可以选择一个按钮,点击一个按钮就释放前面选择的按钮,点击的按钮成为被选择的按钮。

可以用 gtk_radio_button_new_with_label 函数建立带标号的 GtkRadioButton,或者用 gtk_radio_button_new 函数建立不带标号的 GtkRadioButton。建立 GtkRadioButton 只完成工作的一半。因为选择按钮必须和组联系起来,使得任何时候只能选择一个按钮。每当建立选择按钮时,需要用选择按钮和组作为参数调用 gtk_radio_button_group 函数,将选择按钮加到组中去。第一次将选择按钮加到一个组中,应该用 NULL 作为组,因为它还不存在。如果用 NULL 作为组,将建立这个组(事实上,组是一个单链接表[GSList],用来跟踪组中的按钮)。

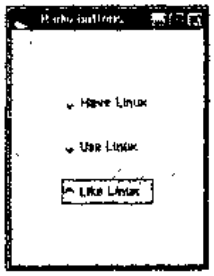


图 4.4 选择按钮

如果在每次将选择按钮增加到组以后,不调用 gtk_radio_button_group 函数取得组(group)将导致不可预期的结果。同样,如果在第一次调用 gtk_radio_button_group 函数前不用 NULL 对组进行初始化通常也会导致程序不正常。

本例说明建立按钮,并将它加到组内。在建立按钮后取得按钮所属的组。

```
GtkWidget *radio;
GSList *group = NULL;

/* --- Create the radio button and add to the group --- */
/* --- if group is NULL, a new group is created --- */
radio = gtk_radio_button_new_with_label (group, szLabel);
```

```
/* Get the group from the button.. The group is a linked list
 * of the button · and the group changes since elements are
 * added to the head. Always get the group after adding button.
 */
group = gtk_radio_button_group (GTK_RADIO_BUTTON (radio));
```

由于选择按钮是由检查按钮派生,而检查按钮又由触发按钮派生,所以选择按钮能够利用触发按钮的事件和函数设置它的属性。由触发按钮使用的事件同样可以由选择按钮使用。

4.6 标号

标号(lebal,GtkLabel)是静态的不可编辑的字段。在屏幕上,常常用标号说明其他字段,在按钮上设置标号用来说明按钮,或者放在其他字段的旁边(如编辑字段)对该字段提供说明。可以用 gtk_label_new 函数建立标号。可以调用 gtk_label_get 函数查询标号的正文,而改变标号的正文需要调用 gtk_label_set 函数。

与其他大多数构件不同,标号不是从 GtkWidget 派生,而是从 GtkMisc 派生的。因此,标号不能和事件联系在一起。标号是轻量级构件,因为建立这一构件时不建立窗口。作为代替,标号显示在父构件中。这种方法降低了开销,不需要跟踪另一个窗口。缺点是这些构件是有限制的。

```
GtkWidget *label;
char *buffer;

/* --- Create a new label --- */
label = gtk_label_new (_This is a label_);

/* --- set the label text to a new value --- */
gtk_label_set (label, _This is the new label_);

/* --- Get the label text --- */
gtk_label_get (label, &buffer);
```

函数 gtk_label_get 引用显示在标号内的正文。如果你需要修改标号的正文,你不应该直接去修改字符串。作为代替,应该使用 gtk_label_set 函数。对标号正文的引用妨碍撤消分配给正文的内存。如果不再使用标号,就不要引用它。

由于标号由 GtkMisc 派生,可以使用 gtk_misc_set_alignment 函数改变正文在标号内的显示方式。函数取 x 对齐值和 y 对齐值来计算正文在标号字段内的显示位置。对 x 的对齐值:0 表示左对齐,.5 表示沿 x 轴居中,1 表示右对齐。对 y 的对齐值:0 对齐顶部,1 对齐底部,.5 表示沿 y 轴居中。

```
/* --- Right aligned, but vertically centered --- */
```

```
gtk_misc_set_alignment (GTK_MISC (label), 1.0, .5);
```

4.7　录入构件

录入构件(entry widget, GtkEntry)是单行录入字段,用它来录入和显示正文信息。录入构件由可编辑构件派生而来,它是更复杂的正文构件的简化形式。录入构件仍然比按钮构件或标号构件复杂。与按钮构件或标号构件相比,它具有有更多的功能。

图 4.5 表明可以从复合框中看到标号和录入构件,在图中看到的选项菜单在本章后面说明。复合框非常类似录入构件,因为复合框使用录入构件和其他构件。



图 4.5　带有标号、录入构件和选项菜单的复合框

可以用 gtk_entry_new 函数或者 gtk_entry_new_with_max_length 函数建立录入构件(GtkEntry)。函数 gtk_entry_new_with_max_length 设置能够录入到构件中的最长的字符数。构件中的正文可以用 gtk_entry_get_text 函数提取。但是,它返回指向字符串数据的指针;不能通过指针修改数据。作为代替,构件中的正文可以用下列 3 种方法之一进行修改。

- 使用 gtk_entry_prepend_text 函数在字段的开始插入正文。
- 使用 gtk_entry_append_text 函数在字段的最后插入正文。
- 使用 gtk_entry_set_text 函数设置字段的正文,但它清除字段的原有值。

在录入构件中可以对光标进行操作。可以用 gtk_entry_select_region 函数查询当前光标(或选择)的位置。还可以用 gtk_entry_set_position 函数将光标移到指定的位置。使用 gtk_entry_select_region 函数可以在构件中选择正文的范围。

录入构件的其他属性包括:是否允许用户修改构件内的正文的指示。可以使用 gtk_entry_set_editable 函数来完成设置构件的可编辑属性。函数 gtk_entry_set_visibility 确定用户能否看到输入字段的正文。在多数情况下,用户应该能够看到正文。但是,如口令字段,就不适合在用户录入时加以显示。

```
/* --- Create an entry field --- */
entry = gtk_entry_new ( );

/* --- Set the default value --- */
gtk_entry_set_text (GTK_ENTRY (entry), ("12:00");
```

```
/* --- Append some text --- */
gtk_entry_append_text (GTK_ENTRY (entry), (" am");

/* --- Don't allow editing of the field --- */
gtk_entry_set_editable (GTK_ENTRY (entry), FALSE);
```

这一录入框能够监听“改变(changed)”信号,它表示改变了录入构件的正文。

4.8 列表框

列表框(list box,GtkList)显示表项的清单。取决于 GtkList 如何配置,用户可以选择一个或多个表项(见图 4.6)。使用 gtk_list_new 函数建立 GtkList。可以用几种方法的一种将表项加到 GtkList 中去。最简单的方法是利用 gtk_list_item_new_with_label 函数和 gtk_container_add 函数的组合。函数 gtk_container_add 能够将多个表项加到 GtkList 中去。因为 GtkList 覆盖了 GtkContainer 的增加功能,它能处理容器中的多个表项。

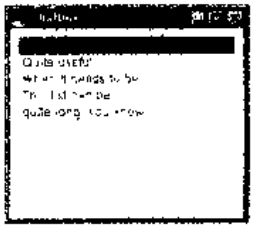


图 4.6 GTKList

下列代码段表明如何建立 GTKList,并将一些表项加到列表框中去。

```
/* --- Create the list box --- */
listbox = gtk_list_new ();

/* --- Allow multiple items to be selected in the list box --- */
gtk_list_set_selection_mode (GTK_LIST (listbox),
                             GTK_SELECTION_MULTIPLE);

/* ---------------------- */
/* --- Add one item --- */
/* ---------------------- */

/* --- Create an item --- */
item = gtk_list_item_new_with_label ("Car");

/* --- Add it to the list --- */
```

```
gtk_container_add (GTK_CONTAINER (listbox), item);
/* ---Make the item visible--- */
gtk_widget_show (item);

/* ------------------------- */
/* ---Add another item--- */
/* ------------------------- */

/* ---Create an item--- */
item = gtk_list_item_new_with_label ("House");

/* ---Add it to the list--- */
gtk_container_add (GTK_CONTAINER (listbox), item);

/* ---Make the item visible--- */
gtk_widget_show (item);
```

当然，当你需要重复使用多个步骤将表项加到列表框中时，最好的做法是将完成增加表项的代码变为一个函数。这种做法简化了代码并使它更容易阅读。可以将上面的程序简化如下：

```
/* ---Create the list box--- */
listbox = gtk_list_new ();

/* ---Allow multiple items to be selected in the list box--- */
gtk_list_set_selection_mode (GTK_LIST (listbox),
                             GTK_SELECTION_MULTIPLE);

/* ------------------------- */
/* ---Add items--- */
/* ------------------------- */

AddListItem (listbox, _Car_);
AddListItem (listbox, _House_);
```

AddListItem 函数具有类似如下的形式：

```
void AddListItem (GtkWidget *listbox, char *sText)
{
    GtkWidget *item;

    /* ---Create list item from data--- */
    item = gtk_list_item_new_with_label (sText);
```

```
    /* --- Add item to listbox --- */
    gtk_container_add (GTK_CONTAINER (listbox), item);

    /* --- Make it visible --- */
    gtk_widget_show (item);
}
```

看上去这一函数并不节省多少工作量。但是如果程序要增加许多表项,它节省的工作量就明显了。如果其他 GtkList 构件使用同一函数增加表项,就能进一步节省工作量。

还可以使用函数将表项的清单加到 GtkList 中去。表项的清单是构件的链接表(GList *),它用 gtk_list_item_new_with_label 函数建立。这些表项能够用 gtk_list_append_items 函数插到 GtkList 的最后;或者用 gtk_list_prepend_items 函数将它们插到 GtkList 的开始;或者用 gtk_list_insert_items 函数插到 GtkList 的特定位置。在每种情况下,Glist 的数据部分是 GtkListItem。

你可以将一组构件加到 GtkList 的最后。函数 CreateListItem(在下面建立)能用来简化编程。这一函数建立表项,使表项成为可视,并返回构件的指针。所以,可以将指针加到链接表中去。

```
/*
 * Create Item
 *
 * Creates a list item with the text passed in and
 * returns it.
 */
GtkWidget *CreateItem (char *sText)
{
    GtkWidget     *item;

    /* --- Create a list item from the data element --- */
    item = gtk_list_item_new_with_label (sText);

    /* --- Show the item --- */
    gtk_widget_show (item);

    /* --- Return the value --- */
    return (item);
```

你可以用 CreatItem 函数建立加到链接表的表项,并用下面的代码建立清单:

```
Items = NULL;
Items = g_list_append (items, CreateItem ("pink"));
Items = g_list_append (items, CreateItem ("blue"));
Items = g_list_append (items, CreateItem ("red"));
```

```
items = g_list_append(items, CreateItem("yellow"));
```

在建立清单后,用下面的方式将它加到 GtkList 的最后:

```
gtk_list_append_items(GTK_LIST(listbox), items);
```

或者用下列方式将 items 加到 GtkList 的开始:

```
gtk_list_prepend_items(GTK_LIST(listbox), items);
```

可以用 gtk_list_clear_items 函数清除 GtkList. 为了清除 GtkList,你必须知道开始和最后的表项的索引。整个清单可以用下列行清除:

```
gtk_list_clear_items(listbox, 0, -1);
```

其中最后的表项设置为 -1 表示清单的末尾。这将清除 GtkList 中的所有表项。

可以用 gtk_list_select_item 函数选择表项,并用 gtk_list_unselect 函数撤消选择。选择/撤消选择的表项取决于传递给函数的索引。除了传递 GtkListItem 代替索引外,函数 gtk_list_select_child 和 gtk_list_unselect_child 完成类似的任务。

```
/* --- Selects child in the list box --- */
gtk_list_select_child(listbox, child);

/* --- Unselects child in the list box --- */
gtk_list_unselect_child(listbox, child);
```

设置列表框的模式(mode)允许在列表框中选择一个或多个表项。可以使用 gtk_list_set_selection_mode 函数来设置列表框的模式。GTK_SELECTION_SINGLE 是最常用的模式,它只允许每次在列表框中选择一个表项。而 GTK_SELECTION_MULTIPLE 模式则允许每次选择多个表项。

在列表框中使用得最多的信号是"selection_changed"("改变了选择")信号。在列表框中,当改变选择的表项或者撤消选择表项时,发出"selection_changed"信号。由于在列表框中的表项是构件,它们也能接受信号;包括"select"信号,它表示表项被选择。在某些情况下,使用"select"信号是方便的,因为能通过事件处理程序给表项传递附加信息。可以对 AddListItem 函数进行修改,使它接受代码和正文,并将它们传递给事件处理程序。

```
/*
 * AddListItem
 *
 * Adds text to the list box.
 */
void AddListItem(GtkWidget *listbox, char *sText, char *sCode)
{
    GtkWidget          *item;
```

```
    /* --- Create a list item from the data element --- */
    item = CreateItem(sText);

    /* --- Add it to the list box --- */
    gtk_container_add(GTK_CONTAINER(listbox), item);

    /* --- Setup signal handler. Note that the code of the item --- */
    /* --- is going to be passed in --- */
    gtk_signal_connect(GTK_OBJECT(item), "select",
                       GTK_SIGNAL_FUNC(listitem_selected), sCode);
}

/*
 * listitem_selected
 *
 * If an item is selected in the list box, the call is made
 * here and the code is passed in as the data parameter.
 */
void listitem_selected(GtkWidget *widget, gpointer *data)
{
    g_print("code of item selected - %s\n", (char *)data);
}
```

4.9 复合框

复合框(combo box,GtkCombo)是编辑字段和列表框的组合(参看图 4.5)。编辑字段允许录入值,或者将值限制在复合框的下移部分(drop-down portion)。复合框可以用 gtk_combo_new 函数建立,并用 gtk_combo_set_popdown_strings 函数将表项放到复合框内。

```
/* ---------------------------- */
/* --- Create a list of the items first --- */
/* ---------------------------- */
cbitems = NULL;
cbitems = g_list_append(cbitems, "Car");
cbitems = g_list_append(cbitems, "House");
cbitems = g_list_append(cbitems, "Job");
cbitems = g_list_append(cbitems, "Computer");

/* --- Make a combo box --- */
combo = gtk_combo_new();
```

```
/* --- Create the drop-down portion of the combo -- */
gtk_combo_set_popdown_strings (GTK_COMBO(combo), cbitems);
```

使用 gtk_combo_set_value_in_list 函数可以限制 GtkCombo 将值放在下移列表框内。用这种方式限制 GtkCombo 并不总是明智的。如果你不希望用户在复合框的编辑部分敲入数据，可以将编辑部分设置为只读。这样，任何人都不能在此字段输入数据。应用程序可以从复合框提取录入数据，并且像访问标准录入字段一样进行访问。下面的程序防止在编辑字段输入数据。

```
/* --- Get the entry field portion of the combo box --- */
entry = GTK_ENTRY (GTK_COMBO (combo)->entry);

/* --- Do not allow it to be edited --- */
gtk_entry_set_editable (entry, FALSE);
```

复合框的录入字段能够捕获"changed"信号，它表示复合框录入字段的值发生了变化。

```
Gtk_signal_connect (GTK_OBJECT (GTK_COMBO (combo)->entry),
                    "changed",
                    GTK_SIGNAL_FUNC (combofunc),
                    NULL);
```

GtkCombo 没有定义函数来清除复合框。但是 GtkCombo 有 GtkWidget，可以用它保持跟踪在下移列表中的表项，并用它来处理 GtkCombo 的表项。因此，清除工作变得简单。只要取得列表框，并用 gtk_list_clear_items 函数就行。

```
GtkList *list;
/* --- Get the list box from the combo --- */
list = GTK_LIST (GTK_COMBO (widget)->list);

/* --- Clear the list within the combo --- */
gtk_list_clear_items (list, 0, -1);

/* --- Clear list in one step --- */
gtk_list_clear_items (GTK_LIST (GTK_COMBO (widget)->list), 0, -1);
```

4.10 选项菜单

选项菜单(option menu，GtkOptionMenu)类似列表框，但它没有编辑功能。虽然两者的功能相似，它们的外观差别很大。选项菜单有时更像复合框(参见图 4.5)。

当点击选项菜单时，出现一个菜单告诉用户这一构件的有效选择范围。选择一个值将导致这一值显示在构件上，同时菜单消失。

首先必须用 gtk_option_menu_new 函数建立 GtkOptionMenu。然后需要建立 GtkRa-

dioMenuItem,并像选择按钮一样将它加到一个组,再将它加到 GtkMenu。再将 GtkMenu 与 GtkOptionMenu 联系在一起。

GtkOptionMenu 的初始化如下:

```
/* --- Create the option menu --- */
omenu = gtk_option_menu_new ();

/* --- Create a menu --- */
menu = gtk_menu_new ();

/* --- No group --- */
group = NULL;
```

通过下列过程将每一项加到选项菜单中。

```
/* --- Value to be displayed --- */
sText = "Medium";

/* --- Create a menu item with a label --- */
menuitem = gtk_radio_menu_item_new_with_label (group, sText);

/* --- Get the group that the menu item is in --- */
group = gtk_radio_menu_item_group (GTK_RADIO_MENU_ITEM (menuitem));

/* --- Add the menu item to the menu --- */
gtk_menu_append (GTK_MENU (menu), menuitem);

/* --- Make the item visible --- */
gtk_widget_show (menuitem);

/* --- Let me know when it's selected and pass in the text --- */
/* --- to the callback function --- */
gtk_signal_connect_object (GTK_OBJECT (menuitem),
                           "activate",
                           GTK_SIGNAL_FUNC (combo_select),
                           (gpointer) sText);
```

"activate"信号表示选择了 GtkRadioMenuItem。将所有表项加到菜单以后,可以用如下方式将菜单和选择菜单联系在一起。

```
/* --- Associate the menu with the option menu --- */
gtk_option_menu_set_menu (GTK_OPTION_MENU (omenu), menu);

/* --- Make it visible --- */
```

```
gtk_widget_show(omenu);
```

选项菜单给用户提供可选择的选项清单，用户选择后就显示当前的选项。

4.11 容器

容器(GtkContainers)是构件。将其他构件放在容器内，容器就成为其他构件的父构件。因为不存在 gtk_container_new 函数，所以不能单独建立容器。但是经常用到容器构件，在前面提到的主窗口、按钮和列表框都是容器构件的例子。容器构件的主要用途是提供通用函数，这些函数可以用在所有容器构件上，不需要为每种构件写一组特殊的函数。

可以使用 gtk_container_add 函数将其他构件加到容器构件内，并可以使用 gtk_container_remove 函数将构件移去。大多数容器构件只允许加一个子构件。有一些例外，如列表框。它专门用来存储多个表项。如果需要在一般容器内存放多个构件，需要用组合表或组合框来保存这些构件。

可以使用 gtk_container_children 函数得到存储在容器内的子构件列表。这一函数返回子构件的链接表(GList *)。另一种方法是使用 foreach 函数。用构件作为参数，函数 gtk_container_foreach 函数对每个子构件完成 foreach 功能。

函数 gtk_container_border_width 控制容器内构件边缘的宽度。将此值设置为 0 使容器内的构件没有边缘。

4.12 小结

基本构件包括建立构件，并在显示构件以前设置构件的属性。构件有设置属性的函数；大多数事件信号都能被应用程序捕获。在本章中讨论了标号、按钮、列表框、录入字段和复合框。这些是在应用程序中经常用到的构件。理解这些构件有助于理解更复杂的构件。

第5章　菜单、工具条和工具提示

应用程序的好的界面和它的内涵一样重要。用户首先看到的是界面。如果界面不能使用户直观和快捷地取得他们所需要的功能，开发人员就没有写好应用程序。正好比销售汽车一样，客户既关心汽车本身的质量，也关心它的营销方式。应用程序的界面和它的内部算法一样重要。非开发人员关心应用程序界面和便于使用的程度远超过对其内部算法的关注。

用户已习惯利用设计良好的菜单来发命令和利用工具条快速访问常用的功能。在本章中，将利用 GTK + 函数设计小型的界面。这一界面包括功能菜单和工具条。

5.1　开始

必须建立应用程序的顶层窗口，并设置它的属性。通常在顶层窗口中设置诸如窗口大小、标题、边缘构件等属性。应用程序的标题是必须设置的表项。使用函数 gtk _ window _ new 建立顶层窗口，并用它来设置应用程序的其他属性。

```
win = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

在建立窗口后，利用 gtk _ window _ new 返回的构件指针作为参数，调用构件处理函数设置窗口的属性。用户在屏幕上利用标题将应用窗口和其他窗口区分开来。应该使用短而说明窗口性质的标题。调用 gtk _ widget _ set _ title 函数设置标题：

```
gtk_widget_set_title (GTK_WINDOW (win), "Application title");
```

如果应用是某种类型的编辑程序，它的标题应该包含文件名称，以此来提醒用户当前编辑的文件是什么。记住：任何时候都可能同时打开几个窗口。标题应该有助于用户确定每一个窗口。

窗口的大小取决于窗口内的构件的大小。窗口的最小尺寸可以用 gtk _ widget _ set _ usize 函数预先设置。有时候通过构件确定的窗口尺寸太小。如果建立的窗口小于窗口的最小尺寸，将把窗口调整到最小尺寸。下列语句将窗口设置为 300 像元乘 200 像元。

```
gtk_widget_set_usize (win, 300, 200);
```

可以用 gtk _ container _ border _ width 函数设置窗口的边缘。下面的例子将容器的边缘设置为 0：

```
gtk_container_border_width (GTK_CONTAINER (win), 0);
```

在第 3 章("开发 GUI 应用")中，讨论了如何使窗口成为可见，以及如何监听"delete _ event"信号。在应用窗口中应该重复这一步。"delete _ event"信号表示窗口即将关闭。在准备好窗口后，可以将 GUI 组成部分加到窗口中去，使应用更具有真实感。

5.2　菜单

手工建立菜单是费事的方法。

在 GTK + 中菜单是构件，它们像其他构件一样运作。人们通常将菜单看成由两部分组成：横跨窗口顶部的菜单条和从菜单条下拉的若干菜单。菜单条通常是应用窗口最上层的构件。菜单条放在窗口的最上层接着放工具条意味着应将菜单放入纵向组合框。在建立菜单条以前必须先建立应用程序的纵向组合框。

```
/* --- Create the application_s vertical packing box --- */
vbox_main = gtk_vbox_new (FALSE, 0);

/* --- Add the packing box to the main window --- */
gtk_container_add (GTK_CONTAINER (win), vbox_main);

/* --- Make the packing box visible --- */
gtk_widget_show (vbox_main);
```

菜单由菜单条(GtkMenuBar)和从菜单条或其他菜单下拉的菜单(GtkMenu)组成。菜单的每一种选择是建立和插入菜单的菜单项(GtkMenuItem)。

第一步是建立菜单条(GtkMenuBar)，并把它加到纵向组合框内。函数 gtk_menu_bar_new 建立 GtkMenuBar 构件。下面是建立菜单条和将它放入纵向组合框的代码：

```
/* --- Create a menu bar for the application --- */
menubar = gtk_menu_bar_new ();

/* --- Put the menu bar in the application packing box --- */
gtk_box_pack_start (GTK_BOX (vbox), menubar, FALSE, TRUE, 0);

/* --- Make the menu bar visible --- */
gtk_widget_show (menubar);
```

上面的代码建立空的 GtkMenuBar。为了使菜单条变为有用，需要增加菜单项。

在建立基本构件时，将若干步骤组合在一个函数内，建立相应的包装函数(wrapper)。建立菜单时可以使用同样的方法。为建立菜单项，第一步是建立带标号的"File"菜单项(GtkMenuItem)。

```
/* --- Create a menu item for "File" --- */
menuFile = gtk_menu_item_new_with_label ("File");
```

在建立菜单项后，需要将它加到前面建立的菜单条中。

```
/* --- Add file to the menu bar --- */
gtk_menu_bar_append (GTK_MENU_BAR (menubar), menuFile);
```

```
/* --- Make the widget visible --- */
gtk_widget_show (menuFile);
```

在大多数应用中,"File"菜单项包含其他菜单项,只能在下拉菜单时才能看到这些菜单项。当选择"File"菜单项时,它并不使应用完成某项操作。作为代替,"File"菜单打开一个子菜单,其中包括诸如"New"、"Open"和"Save"等选项。事实上"File"菜单项导致打开新的菜单,需要将它和包含其他选项的新菜单联系在一起。先建立包含"New"、"Open"和"Save"等选项的新菜单,再用 gtk_menu_item_set_submenu 函数将新菜单和"File"菜单项联系在一起。

```
/* --- Create the new menu for "New," "Open," etc --- */
menu = gtk_menu_new ( );

/* --- Mark the submenu under the "File" menuitem --- */
gtk_menu_item_set_submenu (GTK_MENU_ITEM (menuFile), menu);
```

现在,可以将菜单项加到新的菜单中去。为了简化子菜单的建立过程,应该将建立子菜单的过程变成一个函数。

```
/*
 * CreateBarSubMenu
 *
 * Create a submenu
 *
 * menu - higher level menu
 * szName - Name of submenu item to be added to menu
 */
GtkWidget *CreateBarSubMenu (GtkWidget *menu, char *szName)
{
    GtkWidget *menuitem;
    GtkWidget *submenu;

    /* --- Create menu --- */
    menuitem = gtk_menu_item_new_with_label (szName);

    /* --- Add it to the menu bar --- */
    gtk_menu_bar_append (GTK_MENU_BAR (menu), menuitem);
    gtk_widget_show (menuitem);

    /* --- Get a menu and attach to the menu item --- */
    submenu = gtk_menu_new ();
    gtk_menu_item_set_submenu (GTK_MENU_ITEM (menuitem), submenu);
```

```
    /* --- Viola! --- */
    return (submenu);
}
```

在增加了子菜单后，可以将子菜单选项逐个增加到子菜单中。从建立子菜单选项开始：

```
menuitem = gtk_menu_item_new_with_label ("New");
```

然后将菜单项加到已经建立的子菜单中，并使菜单项成为可见。

```
gtk_menu_append (GTK_MENU (submenu), menuitem);
gtk_widget_show (menuitem);
```

当然，现在还不能用菜单做任何事情。首先，需要捕捉指示用户选择了菜单项的事件。菜单项具有表示菜单项被选择的“activate(激活)”信号。对这一信号建立相应的回呼函数，应用程序就能对用户选择的菜单选项作出响应。

在菜单选项之间可以插入分隔条(separator)。可以调用不带标号的 gtk_menu_item_new 函数建立分隔条，并将它加到子菜单中。这一步在子菜单中将分隔命令组的水平分隔条插入到命令之间。在长的菜单中，分隔条特别有用。必须将“exit(退出)”选项和其他菜单选项分隔开，以避免用户选择与它邻近的菜单项时错误地选择了退出项。

5.2.1　检查菜单项

除了处理命令外，菜单还可以用来显示状态信息。检查菜单项(GtkCheckMenuItem)在菜单中的作用类似检查框，它允许你检查和修改菜单中的布尔值。除了改用 gtk_check_menu_item_new_with_label 函数建立菜单项和需要捕捉“toggled(触发)”信号外，建立 GtkCheckMenuItem 的步骤类似建立普通菜单项。你可以建立单独的函数来完成所有建立检查菜单项的工作

```
/*
 * CreateMenuCheck
 *
 * Create a menu item that has a check box
 *
 * menu - parent menu to add the check menu item
 * szName - name assigned to the menu item
 * func - event handler to call when menu item is toggled
 * data - additional info to pass to the event handler
 */
GtkWidget *CreateMenuCheck(GtkWidget *menu,
                           char *szName,
                           GtkSignalFunc func,
                           gpointer data)
```

```
    GtkWidget *menuitem;

    /* --- Create menu item --- */
    menuitem = gtk_check_menu_item_new_with_label (szName);

    /* --- Add it to the menu --- */
    gtk_menu_append (GTK_MENU (menu), menuitem);
    gtk_widget_show (menuitem);

    /* --- Listen for "toggled" messages --- */
    gtk_signal_connect (GTK_OBJECT (menuitem), "toggled",
                        GTK_SIGNAL_FUNC(func), data);

    return (menuitem);
}
```

检查菜单项可以用 gtk_check_menu_item_set_state 函数设置。

```
/* --- Set it at false --- */
gtk_check_menu_item_set_state (GTK_CHECK_MENU_ITEM (menuitem), FALSE);
```

5.3 选择菜单项

正像菜单项能用作检查框一样,也可以在组内将菜单项用作选择按钮。要将菜单项用作选择按钮,需要用 gtk_radio_menu_item_new_with_label 函数建立菜单项。这一函数取准备填入选择按钮的组名和菜单项名称作为参数。在将第一个选择菜单项加到组内以前,应该将组设置为 NULL;在组内每增加一新项后,需要提取组的新值。下列代码说明建立选择菜单项的过程:

```
GSList *group = NULL;

/* --- Add "Tall" to the group --- */
menuitem = gtk_radio_menu_item_new_with_label (group, "Tall");

/* --- Update the group that we added the item to --- */
group = gtk_radio_menu_item_group (GTK_RADIO_MENU_ITEM (menuitem));
/* --- Add the menu item to a menu --- */
gtk_menu_append (GTK_MENU (menu), menuitem);

/* --- Show the menu item --- */
gtk_widget_show (menuitem);
```

```
/* --- Connect the signal handler --- */
gtk_signal_connect (GTK_OBJECT (menuitem), "toggled",
                    GTK_SIGNAL_FUNC (TallFunc), NULL);
```

选择菜单项(GtkRadioMenuItem)由检查菜单项(GtkCheckMenuItem)派生出来,因此它们使用的信号的相同的。而且,设置和取得选择菜单项状态的函数与设置和取得检查菜单项状态的函数也相同。事实上,选择菜单项可以用下面的代码进行设置:

```
/* --- Set it at FALSE --- */
gtk_check_menu_item_set_state (GTK_CHECK_MENU_ITEM (menuitem), FALSE);
```

5.4　工具提示

可以给用户提供的另一项有用的特征是工具提示(tooltip)。工具提示是一些小的弹出正文的窗口;当将鼠标移到构件上时,将显示相应的窗口;当将鼠标移出构件时,窗口就消失。它们向用户提供有关构件的附加信息;也就是说,工具提示解释这一构件用来做什么。(如果工具条的图案是由不胜任的艺术家绘制的,在这种情况下工具提示就显得特别有用)。在多数情况下,工具提示和工具条一起使用,但也可以与其他构件一起使用。用 gtk_tooltips_new 函数建立工具提示。

```
/* --- Create tooltips --- */
tooltips = gtk_tooltips_new ( );
```

在建立工具提示后,可以使用 gtk_tooltips_set_tip 函数将它们加到菜单项中去:

```
/* --- Add the tip to a menu item --- */
gtk_tooltips_set_tip (tooltips, menuitem,
        "An apple a day keeps the doctor away", NULL);
```

你能够使用 gtk_tooltips_set_colors 函数使提示具有色彩。传递背景(background)和前景(foreground)颜色(GtkColor),可以改变工具提示的颜色。

```
/* --- Change the color --- */
gtk_tooltips_set_colors (tooltips, background, foreground);
```

可以使用 gtk_tooltips_set_delay 函数改变工具提示的延迟时间。使用 gtk_tooltips_disable 函数禁止工具提示的显示,并用 gtk_tooltips_enable 函数恢复工具提示的显示

5.5　快捷键

许多人并不喜欢使用鼠标选择菜单,他们使用快捷键输入命令。优秀的菜单开发人员始终注意到这一点。他们对常用的命令提供快捷键。使用快捷键以前,需要建立快捷键,并将它

们和窗口联系起来。使用 gtk _ accel _ group _ new 函数建立快捷键组(GtkAccelGroup),并用 gtk _ window _ add _ accelerator _ table 函数将快捷键组和窗口联系起来。

```
/* --- Create the accelerator table --- */
accel_group = gtk_accel_group_new();

/* --- Add the accelerator table to the application window --- */
gtk_window_add_accelerator_table(accel_group, GTK_OBJECT(win_main));
```

虽然建立了快捷键组,但还没有将快捷键映射到菜单中去。gtk _ widget _ add _ accelerator 函数将键盘命令映射到菜单中。函数取 GtkMenuItem、快捷键组、信号(通常是"activate")和产生信号使用的键作为参数。下面的例子将 Control - C(GDK _ CONTROL _ MASK, 'C')映射到 GtkMenuItem,并使快捷键在屏幕上成为可见(GTK _ ACCEL _ VISIBLE)。当在键盘上敲入 Control - C 时,将发生"activate"信号。

```
/* --- The Control-C key activates this menu --- */
gtk_widget_install_accelerator(menuitem,
                               "activate",
                               accel_group,
                               'C',
                               GDK_CONTROL_MASK,
                               GTK_ACCEL_VISIBLE);
```

5.6 合并代码

可以将建立 GtkMenuItem 的所有步骤(菜单、快捷键和工具提示)合并成为一个函数。这个函数可以用来代替建立每个菜单项的 10 ~ 15 行代码,并使程序更容易编写。

这一函数有用于工具提示和快捷键的参数。如果不使用工具提示和快捷键,就不必设置这些参数。如果不提供菜单标号的名称,则建立分隔条。整个函数如下所示:

```
/*
 * CreateMenuItem
 *
 * Creates an item and puts it in the menu and returns the item.
 *
 * menu - container menu
 * szName - Name of the menu - NULL for a separator
 * szAccel - Acceleration string - "^C" for Control-C
 * szTip - Tooltips
 * func - Callback function
 * data - Callback function data
 *
```

```
 * returns new menu item
 */
GtkWidget *CreateMenuItem(GtkWidget *menu,
                          char *szName,
                          char *szAccel,
                          char *szTip,
                          GtkSignalFunc func,
                          gpointer data)
{
    GtkWidget *menuitem;

    /* --- If there's a name, create the item and put a
     *     signal handler on it.
     */
    if (szName && strlen(szName)) {
        menuitem = gtk_menu_item_new_with_label(szName);
        gtk_signal_connect(GTK_OBJECT(menuitem), "activate",
                           GTK_SIGNAL_FUNC(func), data);
    } else {
        /* --- Create a separator --- */
        menuitem = gtk_menu_item_new();
    }

    /* --- Add menu item to the menu and show it. --- */
    gtk_menu_append(GTK_MENU(menu), menuitem);
    gtk_widget_show(menuitem);

    if (accel_group == NULL) {
        accel_group = gtk_accel_group_new();
        gtk_accel_group_attach(accel_group, GTK_OBJECT(win_main));
    }

    /* --- If there was an accelerator --- */
    if (szAccel && szAccel[0] == '') {
        gtk_widget_add_accelerator(menuitem,
                                   "activate",
                                   accel_group,
                                   szAccel[1],
                                   GDK_CONTROL_MASK,
                                   GTK_ACCEL_VISIBLE);
    }
```

```
    /* --- If there was a tooltip --- */
    if (szTip && strlen (szTip)) {
        gtk_tooltips_set_tip (tooltips, menuitem, szTip, NULL);

    }

    return (menuitem);
}
```

当有子菜单时,可以用上面的函数建立菜单项。可以使用这一函数建立带工具提示的"New"和"Open"菜单项如下:

```
/* --- Create "new" --- */
menuitem = CreateMenuItem (menu, "new", "N",
                "Create a new item",
                GTK_SIGNAL_FUNC (NewFunc), "new");

/* --- Create "Open" --- */
menuitem = CreateMenuItem (menu, "Open", "O",
                "Open an existing item",
                GTK_SIGNAL_FUNC (OpenFunc), "Open");
```

用单独的函数使程序更容易阅读。由于建立菜单的代码集中在一起,也便于以后对程序进行修改。可以对函数作进一步的改进,将数据保存在结构中,并从结构中读取数据。这一步留给读者作为练习。

5.7 套件

使用套件(item factory,GtkItemFactoryEntry)是建立菜单的方便的方法。套件使用预先填充的结构作为菜单的基础。结构包括 5 个部分:菜单路径、快捷键、回呼函数、回呼函数的附加参数和标志。标志用来指示菜单项是什么。标志可以取下列值:

- <Title> 建立标题项。
- <Item> 建立简单的菜单项。
- <Check> 建立检查菜单项。
- <ToggleItem> 建立触发菜单项。
- <RadioItem> 建立选择菜单项。
- <Separater> 建立分隔项。
- <Branch> 建立保存子菜单的菜单项。
- <LastBranch> 建立右对齐的保存子菜单的菜单项。

如果只建立简单的菜单项,可以使用 NULL、0 或""作为标志。

例如,在"File"菜单中的"New"子菜单可以用下列行建立:

```
"/File/New", "<control>N", FileNew, NULL, "0"
```

"/File/New"用来确定路径。在本例中,第一个"/"表示根菜单或菜单条。"File"是菜单条中的菜单项,"New"是"File"菜单的菜单项。"<control>N"表示调用这一菜单的快捷键是 Ctrl + N。快捷键可以用<shift>序列(如"<shift>b")表示按下 shift 键,也可以用<alt>序列(如"<alt>q")表示按下 alt 键。回呼函数是标准的回呼函数。当用户用鼠标选择菜单项或从键盘敲入快捷键时,将调用回呼函数,并传送额外的字段。将它设置为 NULL 将传送 NULL。套件的回呼函数在参数的传递上稍有不同。调用回呼函数时,按顺序使用从表、回呼函数操作以及构件得到的回呼函数数据。

5.7.1　对套件进行编程

一旦建立了 GtkItemFactoryEntry 表,使用表的程序不会很长。第一步是用 gtk_item_factory_new 函数建立 GtkItemFactory;建立套件时用 GTK_TYPE_MENU_BAR 作为类型参数,并用名称和由 gtk_accel_group_new 函数建立的快捷键组作为参数。使用 gtk_item_factory_create_items 函数将 GtkItemFactoryEntry 的菜单项加进去,并传入表的信息。下面的例子用套件建立窗口的菜单。这一子程序实现显示操作以及实现退出应用程序的 Quit 操作的回呼函数。

```
/*
 * menu.c
 *
 * Menus using the item factory
 */

#include <gtk/gtk.h>

static void QuitApp (gpointer callback_data,
                     guint callback_action,
                     GtkWidget *widget);
static void ShowMenu (gpointer callback_data,
                     guint callback_action,
                     GtkWidget *widget);

/*
 * Structure to build the menus
 */
static GtkItemFactoryEntry menu_items[] = {
  {"/_File",            NULL,         0,          0, "<Branch>"  },
  {"/File/tearoff1",    NULL,         ShowMenu,   0, "<Tearoff>" },
  {"/File/_New",        "<control>N", ShowMenu,   0 },
  {"/File/_Open",       "<control>O", ShowMenu,   0 },
```

```
    {"/File/_Save",             "<control>S",   ShowMenu,       0},
    {"/File/Save _As...",       NULL,           ShowMenu,       0},
    {"/File/sep1",              NULL,           ShowMenu,       0, "<Separator>"},
    {"/File/_Quit",             "<control>Q",   QuitApp,        0},

    {"/_Edit",                  NULL,           0,              0, "<Branch>"},
    {"/_Edit/Cut",              "<control>X",   0,              0, 0},
    {"/_Edit/_Copy",            "<control>C",   0,              0, 0},
    {"/_Edit/_Paste",           "<control>V",   0,              0, 0},
    {"/_Edit/_Font",            NULL,           0,              0, "<Branch>"},
    {"/_Edit/Font/_Bold",       NULL,           ShowMenu,       0, "<RadioItem>"},
    {"/_Edit/Font/_Italics",    NULL,           ShowMenu,       0, "<RadioItem>"},
    {"/_Edit/Font/_Underline",  NULL,           ShowMenu,       0, "<RadioItem>"},
    {"/_Edit/_Color",           NULL,           0,              0, "<Branch>"},
    {"/_Edit/Color/_Red",       NULL,           ShowMenu,       0, "<CheckItem>"},
    {"/_Edit/Color/_Blue",      NULL,           ShowMenu,       0, "<CheckItem>"},
    {"/_Edit/Color/_Green",     NULL,           ShowMenu,       0, "<CheckItem>"},

    {"/_Help",                  NULL,           0,              0, "<LastBranch>"},
    {"/Help/_About",            NULL,           ShowMenu, 0},
};

/*
 * FactoryQuit
 *
 * Quit the app when called from the menu.
 */
static void QuitApp (gpointer callback_data,
                     guint callback_action,
                     GtkWidget *widget)
{
    /* --- Display a message about the menu --- */
    g_message ("ItemFactory: activated \"%s\"",
               gtk_item_factory_path_from_widget (widget));

    /* --- Exit the application --- */
    gtk_main_quit ();
}

/*
 * ShowMenu
 *
```

```
    * Display the item that was selected
    */
static void ShowMenu (gpointer callback_data,
                      guint     callback_action,
                      GtkWidget *widget)
{
    g_message ("ItemFactory: activated - \"%s\", action %d",
               gtk_item_factory_path_from_widget (widget), (int) callback_action);
}

/*
 * --- Number of items in the menu
 */
static int nmenu_items = sizeof (menu_items) / sizeof (menu_items[0]);

/*
 * CloseApp
 *
 * Close down GTK when they close the application
 * window
 */
static gint CloseApp (GtkWidget *widget, gpointer data)
{
    gtk_main_quit ();
    return (TRUE);
}

/*
 *
 */
static void CreateItemFactory ()
{
    GtkWidget *window = NULL;
    GtkWidget *box1;
    GtkWidget *box2;
    GtkWidget *separator;
    GtkWidget *label;
    GtkWidget *button;
    GtkAccelGroup *accel_group;
    GtkItemFactory *item_factory;

    /* --- Create the window --- */
```

```
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

/* --- Set the minimum size --- */
gtk_widget_set_usize (window, 200, 200);

/*  --Connect signals for destruction --- */
gtk_signal_connect (GTK_OBJECT (window), "destroy",
                    GTK_SIGNAL_FUNC (gtk_widget_destroyed),
                    &window);
gtk_signal_connect (GTK_OBJECT (window), "delete-event",
                    GTK_SIGNAL_FUNC (CloseApp),
                    NULL);

/* --- Create a new accel group --- */
accel_group = gtk_accel_group_new ();

/* --- Create a new item factory -    */
item_factory = gtk_item_factory_new (GTK_TYPE_MENU_BAR,
                                     "<blah>",
                                     accel_group);

/* --- Create the items in the factory using data --- */
gtk_item_factory_create_items (item_factory,
                               nmenu_items,
                               menu_items,
                               NULL);

/* --- Attach the accel group to app window --- */
gtk_accel_group_attach (accel_group, GTK_OBJECT (window));

/* --- Set the title of the window --- */
gtk_window_set_title (GTK_WINDOW (window), "Item Factory");

/* --- No border --- */
gtk_container_border_width (GTK_CONTAINER (window), 0);

/* --- Vertical packing box. --- */
box1 = gtk_vbox_new (FALSE, 0);
gtk_container_add (GTK_CONTAINER (window), box1);

/* --- Put menu in the vertical packing box --- */
gtk_box_pack_start (GTK_BOX (box1),
```

```
                gtk_item_factory_get_widget (item_factory, "<blah>"),
                FALSE, FALSE, 0);

    /* ---Make everything visible --- */
    gtk_widget_show_all (window);
}

int main (int argc, char *argv[])
{

    gtk_init (&argc, &argv);

    CreateItemFactory ();

    gtk_main ();

    return 0;
}
```

5.8　使用套件和手工建立菜单的比较

套件是快速建立菜单的好方法，但使用它们建立菜单有一定限制。如不能用它们在菜单中增加诸如像元图一类的构件。然而，对大多数应用讲，使用套件是将菜单加到应用中去的快速和简单的方法。手工建立菜单更复杂，但也更为灵活。

5.9　像元图

在按钮或其他构件上显示图像需要像元图(pixmap)。像元图是影像构件，经常用在按钮或其他构件上作为图标，以图像的形式说明按钮或构件的用途。在应用中的工具条有一整行图形按钮。它们比基于正文的按钮占用更少的空间，而且使用户以更快的方式找到工具条的命令。建立像元图的一种方式是从图形编辑软件(如 GIMP)中建立它，并将它作为 xmpFile 保存。下面是用 GIMP 建立 xmpFile 的例子(包括注释)：

```
static const gchar *xpm_open[] = {
"16 16 4 1",              /* 16 x 16 bitmap, 4 colors, 1 char/color */
"  c None",               /* color 1 - ' ' = transparent color */
"B c #000000000000",      /* color 2 - 'B' = black */
"Y c #FFFFFFFF0000",      /* color 3 - 'Y' = yellow */
"Y c #999999990000",      /* color 4 - 'Y' = dark yellow */
"                ",       /* image data begins here */
```

```
"            BBB    ",
"  BBBBBB  B   BB  ",
"  BYYYB       BB  ",
" BYYYYYBBBBBB      ",
" BYYYYYYYYYB       ",
" BYYYYYYYYYB       ",
" BYYYYYYYYYB       ",
" BYYBBBBBBBBBB     ",
" BYYByyyyyyyyyB     ",
" BYByyyyyyyyyB     ",
" BYByyyyyyyyyB     ",
" BByyyyyyyyyB      ",
" BByyyyyyyyyB      ",
" BBBBBBBBBBB       ",
"                   ",    /* image data ends here */
};
```

xmpFile 由 3 部分组成:header,彩色表和像元图数据。Header 是数组中的第一个字符串,字符串类似"16 16 4 1",其中前面两个数字表示影像的高度和宽度。第 3 个数字表示在字符串数组中定义的颜色数目,第 4 个数字表示每种颜色的字符数。在本例中,"16 16 4 1"表示影像高 16 个像元,宽 16 个像元. 第 3 个数表示在 header 后面构成颜色数据的字符串数. 在本例中,有 4 行颜色数据:

```
"  c None"
"B c #000000000000"
"Y c #FFFFFFFF0000"
"y c #999999990000"
```

每一颜色数据行定义影像中的一种颜色。header 的第 4 个数字指定用多少个字符(通常用 1 个字符)定义一种颜色。

每一颜色数据行定义一个字符(如字符 c)和代表颜色的 16 进制值(如 None 表示透明)。颜色值用 RGB 格式表示。在上面的例子中,空表示无彩色;B 表示黑(#000000000000);Y 表示黄(#FFFFFFFF0000);y 表示深黄(#999999990000)。这里表示颜色的 16 进制值是 8 字节的颜色值。可以用 4 字节值来代替它。如深黄用 #999900 代替 #999999990000。

颜色数据的后面是影像数据。使用已经定义的颜色数据,影像可以看作"打开"的图标,类似稍稍打开的黄色的 File 夹。空通常表示不用彩色,它使代码中影像的轮廓更清晰。

定义影像只是开始。为了使用影像需要将它转换为构件,使你能够像处理其他构件一样处理影像。可以建立一个函数取得这些数据,并建立像元图构件。为了建立像元图构件,需要将 GtkPixmap 和掩码(mask)传递给 gtk _ pixmap _ new 函数。幸运的是,可以调用 gtk _ pixmap _ creat _ from _ xpm _ d 函数,它取 xpm 数据结构,并提供掩码和 GtkPixmap。

```
/*
```

```
 * CreateWidgetFromXpm.
 *
 * Convert an xpm string into a pixmap widget
 * Uses (win_main) which is a global.
 */
GtkWidget *CreateWidgetFromXpm(GtkWidget *window, gchar **xpm_data)
{
    GdkBitmap *mask;
    GdkPixmap *pixmap_data;
    GtkWidget *pixmap_widget;

    /* --- Convert the string to a gdk_pixmap --- */
    pixmap_data = gdk_pixmap_create_from_xpm_d(
                                    win_main->window,
                                    &mask,
                                    NULL,
                                    (gchar **)xpm_data);

    /* --- Convert the pixmap to a pixmap widget --- */
    pixmap_widget = gtk_pixmap_new(pixmap_data, mask);

    /* --- Make the widget visible --- */
    gtk_widget_show(pixmap_widget);

    /* --- Return it so it can be used --- */
    return(pixmap_widget);
}
```

这段代码建立像元图构件；当将它加到容器内时就能看到它。你也能建立按钮并将像元图加到按钮上，就得到带有像元图的按钮。这一技术使你可以将图标加到工具条的按钮上。

5.10　工具条

工具条为用户提供常用命令的简便菜单。工具条通常包含带有表示命令的图标的按钮，但也能包含其他构件。使用 gtk_toolbar_new 函数建立工具条，函数取两个参数用来确定工具条的式样。第一个参数用来确定图标排列的方向：纵向排列(GTK_ORIENTATION_HORIZONTAL)或横向排列(GTK_ORIENTATION_VERTCAL)。第二个参数确定按钮显示图标(GTK_TOOLBAR_ICONS)、正文(GTK_TOOLBAR_TEXT)还是同时显示两者(GTK_TOOLBAR_BOTH)。在本例中，它是横贯主窗口的带图标的工具条。需要将工具条加到早先为应用建立的纵向组合框中。

```
/* --- Create a horizontal toolbar with icons --- */
toolbar = gtk_toolbar_new (GTK_ORIENTATION_HORIZONTAL,
                           GTK_TOOLBAR_ICONS);

/* --- Add the toolbar to the application vertical  ·   */
/* --- packing box. This is right under the menu --- */
gtk_box_pack_start (GTK_BOX (vbox), toolbar, FALSE, TRUE, 0);

/* --- Make the toolbar visible --- */
gtk_widget_show (toolbar);
```

现在你有了还没有按钮的工具条。将构件加到工具条和格式化工具条需要用到一个或几个函数。三个常用的函数是:gtk _ toolbar _ append _ item,gtk _ toolbar _ append _ element 和 gtk _ toolbar _ append _ space。

5.10.1 增加工具条按钮

函数 gtk _ toolbar _ append _ item 每次将一个按钮加到工具条中。这一函数使用的参数有:按钮的像元图、工具提示、回呼函数等。函数的原型是:

```
GtkWidget *gtk_toolbar_append_item (
                GtkToolbar *toolbar,
                const char *text,
                const char *tooltip_text,
                const char *tooltip_private_text,
                GtkWidget *widget,
                GtkSignalFunc callback,
                gpointer userdata);
```

其中 toolbar 是待加入按钮的工具条。text 用于正文按钮。这一工具条被定义为图形按钮,所以忽略这一字段。tooltip _ text 定义工具提示正文,当用户将鼠标移动到按钮上时显示这一正文。现在可以不管 tooltip _ private _ text。widget 通常是放在按钮上的像元图构件。Callback 是回呼函数,当按下按钮时调用回呼函数。Userdata 是传递给回呼函数的附加参数。

你可以使用早先建立的 CreateWidgetFromXpm 函数从 xpm 数据建立构件,进一步建立带图标的按钮。传递像元图将把像元图加到按钮上。

```
/* --- Create the "new" button on the toolbar --- */
gtk_toolbar_append_item (GTK_TOOLBAR (toolbar),
            NULL, "New window", NULL,
            CreateWidgetFromXpm (vbox_main, (gchar ** ) xpm_new),
            (GtkSignalFunc) ButtonEvent,
            NULL);
```

按钮显示送入的像元图作为图标。当按下工具条上的按钮时,调用 ButtonEvent 函数。

你可以编写这一函数的代码，使它完成相应的处理。这里使用的标准方法是对同一事件调用同一函数作为"New window"菜单命令。函数 gtk _ toolbar _ append _ item 的优点在于它还完成事件处理程序和工具提示"New window"的设置，不需要任何附加的调用。当你用手工建立菜单时，每一步都要自己去做；所以需要建立函数来做这件事。这里只用一个函数，而且便于使用

5.10.2　在工具条中增加其他项

函数 gtk _ toolbar _ append _ element 类似函数 gtk _ toolbar _ append _ item，但它更为灵活因而也更复杂。它不仅可以用于单纯的工具条按钮，函数 gtk _ toolbar _ append _ element 还支持构件(GTK _ TOOLBAR _ CHILD _ WIDGET)、按钮(GTK _ TOOLBAR _ CHILD _ BUTTON)、触发按钮(GTK _ TOOLBAR _ CHILD _ TOGGLEBUTTON)、选择按钮(GTK _ TOOLBAR _ CHILD _ RADIOBUTTON)和在逻辑构件组之间产生空隙(GTK _ TOOLBAR _ CHILD _ SPACE)。这一函数增加了两个参数：类型和构件。类型参数确定插入工具条的构件的类型。如果类型设置为 GTK _ TOOLBAR _ CHILD _ WIDGET，构件参数则用作插入的构件。

```
GtkWidget *gtk_toolbar_append_element (
                    GtkToolbarChildType type,
                    GtkWidget *widget,
                    GtkToolbar *toolbar,
                    const char *text,
                    const char *tooltip_text,
                    const char *tooltip_private_text,
                    GtkWidget *icon,
                    GtkSignalFunc callback,
                    gpointer userdata);
```

例如，增加触发构件需要用到 gtk _ toolbar _ append _ element 函数。

```
/* --- Create a bold button on the toolbar --- */
tool_bold = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar),
                GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                NULL,
                NULL, "Bold", NULL,
                CreateWidgetFromXpm (vbox_main,
                                     (gchar **) xpm_bold),
                                     (GtkSignalFunc) BoldClicked,
                                     NULL);
```

作为代用方法，可以建立 GtkToggleButton，将它作为构件参数，并将类型参数设置为 GTK _ TOOLBAR _ CHILD _ WIDGET。这就告诉 GTK + 你传送的是定制的构件，而不是建立新构件。这样做的结果和将类型参数设置为 GTK _ TOOLBAR _ CHILD _ TOGGLEBUT-

TON 的效果相同。

这一函数的更简短的形式是 gtk _ toolbar _ append _ widget。它需要更多的设置，也提供更大的灵活性。这一函数定义如下：

```
void gtk_toolbar_append_widget (GtkToolbar **toolbar,
                                GtkWidget *widget,
                                Const gchar *tooltip_text,
                                Const gchar *tooltip_private_text);
```

由于这一函数没有设置回呼函数或图标，这些项的设置需要自己编程来解决。这一方式增加了你需要编写的代码量，但更灵活。当在框外增加构件时，需要这样做。

5.10.3 在工具条中增加空隙

函数 gtk _ toolbar _ append _ space 在工具条结束处增加空隙。函数需要的唯一参数是需要增加空隙的工具条。

```
/* --- Add a gap between widget --- */
gtk_toolbar_append_space (toolbar);
```

函数 gtk _ toolbar _ append _ space 在最后的按钮后面插入空隙。插入空隙有助于将按钮组成逻辑组。对按钮进行逻辑分组(如字体按钮组，File 操作按钮组等)方便用户选择按钮完成特定的任务。

除了从工具条后面增加构件的函数外，还有从工具条前面增加构件的函数。这些函数有 gtk _ toolbar _ prepend _ item、gtk _ toolbar _ prepend _ element、gtk _ toolbar _ prepend _ space 和 gtk _ toolbar _ prepend _ widget。这些函数使用的参数和 append 类函数使用的参数相同。

有时需要在工具条的构件组中间插入构件。通常在改变已经存在的工具条时需要这样做。使用插入(insert)函数，可以将按钮/构件插入工具条的任何位置。插入函数类似 append 和 prepend 函数，但它需要增加一个用来确定插入按钮/构件的位置的参数。增加的参数是表示插入位置的整数索引。传递参数 0 表示插在工具条的最前面；传递参数 toolbar- > num _ childen 表示插入工具条的最后面(toolbar- > num _ childen 表示当前在工具条中按钮/构件的项数)。

5.11 建立应用的用户界面

掌握了前面的基础以后，可以建立用户界面的样本。对它进行适当修改，就能用于所有类型的应用。在建立应用的过程中，如有必要应该合并函数以简化程序。如果你经常调用 3 到 4 个 GTK + 函数来完成一项操作，应该将这些步骤合并为一个函数。例如，在建立菜单的过程中，可以将建立菜单所需要的许多步骤合并为单独的函数。

现在已有条件从头开始建立带有用户界面的应用。本节将讨论如何快速和方便地实现这一目标。你需要建立两个菜单："File"菜单包含"New"、"Open"、"Save"、"Save As…"、分隔项和"Quit"。"Edit"菜单包含"Cut"、"Copy"、"Paste"和"Font"子菜单；后者包含触发按钮项

"Bold"、" Italics"和"Underline"。

5.11.1　建立应用的窗口和菜单

主程序很短。它的主要任务是建立工具提示，并调用 CreateMainWindow 函数来完成主要的工作。

```
/*
 * main
 *
 * --- Program begins here
 */
int main(int argc, char *argv[])
{
    /* --- Initialize GTK --- */
    gtk_init (&argc, &argv);

    /* --- Initialize tooltips --- */
    tooltips = gtk_tooltips_new();

    /* --- Create application --- */
    CreateMainWindow ();

    /* --- Let GTK do its processing --- */
    gtk_main();

    return 0;
}
```

CreateMainWindow 函数完成大多数构件的建立任务，并用很少的步骤完成这些任务。第一部分建立窗口，并设置主窗口的属性。在使窗口成为可视以前，先设置类似窗口的标题和尺寸等属性。对快捷键表进行初始化，并将它指定给主窗口，使你能使用菜单中的快捷键。增加事件处理程序监听"delete_event"信号。在建立窗口以后，加入纵向组合框。使得菜单可以堆积在最上面，将其他东西放在下面。将组合框加入主窗口后，然后使主窗口成为可视。

```
/*
 * CreateMainWindow
 *
 * Create the main window and the menu/toolbar associated with it
 */
Static void CreateMainWindow ()
{
        GtkWidget *vbox_main;
```

```
GtkWidget *menubar;
GtkWidget *menu;
GtkWidget *menuitem;
GtkWidget *menufont;
GtkWidget *toolbar;
GtkWidget *button;

/* --- Create the top window and size it --- */
win_main = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_widget_set_usize(win_main, 360, 260);

/* --- Set the title and border width --- */
gtk_window_set_title (GTK_WINDOW (win_main), "Menu test");
gtk_container_border_width (GTK_CONTAINER (win_main), 0);

/* --- Create accel table --- */
accelerator_table = gtk_accelerator_table_new();
gtk_window_add_accelerator_table(GTK_WINDOW(win_main), accelerator_table);

/* --- Top-level window should listen for the "delete_event" --- */
gtk_signal_connect (GTK_OBJECT (win_main), "delete_event",
        GTK_SIGNAL_FUNC(EndProgram), NULL);

/* --- Create v-box for menu, toolbar --- */
vbox_main = gtk_vbox_new (FALSE, 0);

/* --- Put up v-box --- */
gtk_container_add (GTK_CONTAINER (win_main), vbox_main);
gtk_widget_show (vbox_main);
gtk_widget_show (win_main);

/* --- Menu bar --- */
menubar = gtk_menu_bar_new ();
gtk_box_pack_start (GTK_BOX (vbox_main), menubar, FALSE, TRUE, 0);
gtk_widget_show (menubar);
```

将“File”子菜单加到菜单条,然后将“File”子菜单下的各个菜单项加进去。在这里,按顺序加入“New”、“Open”、“Save”、“Save As…”、分隔项和“Quit”。在整个处理过程中,你需要对每一菜单项提供事件处理函数和工具提示;在多数情况下,同时提供快捷键。对于大多数函数,将回呼函数设置为 PrintFunc 函数。当菜单被选择时,这一函数在控制台显示菜单被选择的消息。如果这个例子是一个应用程序,回呼函数将完成某些其他的操作。最后送给 CreateMenuItem 的参数是数据,当 PrintFunc 函数被调用时,送给它作为参数。这种方式使得许多

菜单项都能利用同一事件处理函数。

```
/* ----------------
  * --- File menu ---
  * ---------------- */
menu = CreateBarSubMenu (menubar, "File");
/* --- _ New _ menu item --- */
menuitem = CreateMenuItem (menu, "new","N",
                "Create a new item",
                GTK_ SIGNAL_ FUNC (PrintFunc), "new");
/* --- _ Open _ menu item --- */
menuitem = CreateMenuItem (menu, "Open","O",
                          "Open an existing item",
                          GTK_ SIGNAL_ FUNC (PrintFunc), "open");
/* --- _ Save _ menu item --- */
menuitem = CreateMenuItem (menu, "Save", "S",
                           "Save current item",
                           GTK_ SIGNAL_ FUNC (PrintFunc), "Save s");
/* --- _ Save as __ menu item --- */
menuitem = CreateMenuItem (menu, "Save As...", " ",
                           "Save current item with new name",
                           GTK_ SIGNAL_ FUNC (PrintFunc), "Save as");
/* --- Separator --- */
menuitem = CreateMenuItem (menu, NULL, NULL,
                NULL, NULL, NULL);
/* --- Quit --- */
menuitem = CreateMenuItem (menu, "Quit", " ",
                "What's more descriptive than quit?",
                GTK_ SIGNAL_ FUNC (PrintFunc), "quit");
```

"Edit"菜单比"File"菜单小，它包含以下各菜单项："Cut"、"Copy"、"Paste"和"Font"。"Font"菜单项实际是一个子菜单，建立"Font"菜单项的函数不同于建立其他菜单项。

```
/* ----------------
  * --- Edit menu ---
  * ---------------- */
menu = CreateBarSubMenu (menubar, "Edit");

/* --- Cut menu item --- */
menuitem = CreateMenuItem (menu, "cut","X",
                "Remove item and place into clipboard",
                GTK_ SIGNAL_ FUNC (PrintFunc), "cut");
```

```
    /* --- Copy menu item --- */
    menuitem = CreateMenuItem (menu,"copy", "^C",
                    "place a copy of the item in the clipboard",
                    GTK_SIGNAL_FUNC (PrintFunc), "copy");
    /* --- Paste menu item --- */
    menuitem = CreateMenuItem (menu, "Paste", "^V",
                    "Paste item",
                    GTK_SIGNAL_FUNC (PrintFunc), "Paste");
```

最后,增加"Font"的菜单项:"Bold"、" Italics"和"Underline"。这些都是检查菜单项,所以在菜单中显示为"被选择"或"未被选择"。在建立所有菜单项后建立工具条。和建立前面一组按钮不同,现在你需要跟踪工具条的构件。将在后面说明原因。

```
    /* ----------------
     * --- Font sub menu ---
     * ---------------- */
    menufont = CreateSubMenu (menu, "Font");

    menuBold = CreateMenuCheck (menufont,
        "Bold", GTK_SIGNAL_FUNC (SelectMenu), "Bold");

    menuItalics = CreateMenuCheck (menufont,
         "Italics", GTK_SIGNAL_FUNC (SelectMenu), "Italics");

    menuUnderline = CreateMenuCheck (menufont,
        "Underline", GTK_SIGNAL_FUNC (SelectMenu), "Underline");

    /* --- Create the toolbar --- */
    CreateToolbar (vbox_main);
}
```

注意,由于在菜单中有检查框,所以在本例中没有用上套件。如果没有检查框,使用套件就比较方便。

5.11.2 建立工具条

函数 CreateToolbar 建立所有带有图标的按钮和工具条。作为附加的内容,在工具条上的字体复合框使它具有专业的外观。首先需要建立工具条,并将它加到纵向组合框中。

```
/*
 * CreateToolbar
 */
void CreateToolbar (GtkWidget * vbox_main)
```

```
{
    GtkWidget *widget;
    /* --- Create the toolbar and add it to the window --- */
    toolbar = gtk_toolbar_new (GTK_ORIENTATION_HORIZONTAL, GTK_TOOLBAR_ICONS);
    gtk_box_pack_start (GTK_BOX (vbox_main), toolbar, FALSE, TRUE, 0);
    gtk_widget_show (toolbar);

    /* --- Create "new" button --- */
    gtk_toolbar_append_item (GTK_TOOLBAR (toolbar),
                             NULL,"new window", NULL,
                             CreateWidgetFromXpm (vbox_main, (gchar ** ) xpm_new),
                             (GtkSignalFunc) ButtonClicked,
                             NULL);

  /* --- Create "open" button --- */
  gtk_toolbar_append_item (GTK_TOOLBAR (toolbar),
                           "Open Dialog","Open dialog", " ",
                           CreateWidgetFromXpm (vbox_main, (gchar ** )xpm_open),
                           (GtkSignalFunc) ButtonClicked,
                           NULL);

  /* --- Little gap --- */
  gtk_toolbar_append_space (GTK_TOOLBAR (toolbar));

  /* --- The_Cut_button --- */
  gtk_toolbar_append_item (GTK_TOOLBAR (toolbar),
                           "Cut","Cut", " ",
                           CreateWidgetFromXpm (vbox_main, (gchar ** )xpm_cut),
                           (GtkSignalFunc) ButtonClicked,
                           NULL);

  /* --- The_Paste_button --- */
  gtk_toolbar_append_item (GTK_TOOLBAR (toolbar),
                           "Paste", "Paste", " ",
                           CreateWidgetFromXpm (vbox_main, (gchar ** )xpm_copy),
                           (GtkSignalFunc) ButtonClicked,
                           NULL);

  /* --- Add a gap --- */
  gtk_toolbar_append_space (GTK_TOOLBAR (toolbar));

  /* --- Create a font combo box. The CreateCombobox
```

```
 *      function creates the combo box and puts an event
 *      handler on the combo box. */
widget = CreateCombobox();

/* --- Add the combo box to the toolbar --- */
gtk_toolbar_append_widget (GTK_TOOLBAR (toolbar),
                           widget,
                           "Font","Pick a font");

/* --- A little gap --- */
gtk_toolbar_append_space (GTK_TOOLBAR (toolbar));

 /*
  * --- Create a toggle button for the Bold
  */
tool_bold = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar),
                GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                NULL,
                NULL, "bold", NULL,
                CreateWidgetFromXpm (vbox_main,
                                     (gchar **) xpm_bold),
                (GtkSignalFunc) ButtonClicked,
                "bold");

/*
 * --- Create a toggle button for the Italics
 */
tool_italics = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar),
                GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                NULL,
                "Italics", "Italics", "Italics",
                CreateWidgetFromXpm (vbox_main,
                                     (gchar * *) xpm_italics),
                (GtkSignalFunc) ButtonClicked,
                "Italics");

/*
 * --- Create a toggle button for the Underline
 */
tool_underline = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar),
                GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                NULL,
                "Underline","Underline","Underline",
                CreateWidgetFromXpm (vbox_main,
```

```
                                    (gchar ** ) xpm_underline),
                ((GtkSignalFunc) ButtonClicked,
                "underline");

}
```

5.11.3　协调工具条和菜单项

在 GTK+ 中，工具条的字体按钮和字体菜单中的选项互不相关。除非在程序中采取措施，否则这两者之间是没有任何联系。建立联系的方式通常是使菜单项和工具条按钮使用同一个回呼函数。这将使应用中的菜单项和工具条按钮产生相同的效果。

在某些应用中，在一些位置需要显示状态信息。在现在正在建立的应用中，字体信息既显示在工具条中，也显示在检查菜单项中。如果在工具条中选择了“Bold”按钮，“Bold”菜单选项同样应该反映状态的改变。然而，在工具条的触发按钮和菜单的触发选项之间的协调并不是自动的　原因很清楚，应用程序并不知道两者之间的关系。

为了实现两者之间必要的协调，使人想起前面将工具条的字体按钮的回呼函数设置为 ButtonClicked 函数。这一函数知道按下了哪一个按钮，并改变相同的菜单项的状态。在建立菜单项时，保存了字体菜单项构件的指针。这使得当选择工具条按钮时改变菜单项的工作变得简单　你可以用下列代码检查触发按钮的触发状态：

```
/* --- Is the button up or down? --- */
nState = GTK_TOGGLE_BUTTON (widget)->active;
```

构件被转换为触发按钮，所以可以检查按钮的状态。可以建立如下的 ButtonClicked 函数来协调两者的状态。

```
/*
 * --- ButtonClicked
 */
void ButtonClicked (GtkWidget *widget, gpointer data)
{
    /* --- Get the button state --- */
    int nState = GTK_TOGGLE_BUTTON (widget)->active;

    /* --- Synchronize the menu state with the button --- */
    SetMenuButton ((char *) data, nState);
}
```

在建立构件时设置了 data 变量，所以可以将它的值传递给函数。在本例中，data 变量告诉你按下了哪一个按钮。回忆一下，你可以使用 gtk_check_menu_item_set_state 函数设置检查按钮的状态。被选择的按钮传递 data 参数，这一参数带有说明按下了哪一个按钮的正文。

```
/*
```

```
 * SetMenuButton
 *
 * Set the menu button based on the name and state.
 *
 * szButton _ name of the button to change
 * nState _ State to set the button
 */
void SetMenuButton (char * szButton, int nState)
{
    GtkCheckMenuItem * check = NULL;

    /* --- Show how we're changing the button --- */
    printf ("check _ menu _ set _ state - %d \n", nState);

    /* --- Bold button? --- */
    if (!strcmp (szButton, "bold")) {

        check = GTK _ CHECK _ MENU _ ITEM(menuBold);

    /* --- Italic button? --- */
    } else if (!strcmp (szButton, "italics")) {

        check = GTK _ CHECK _ MENU _ ITEM(menuItalics);

    /* --- Underline button? --- */
    } else if (!strcmp (szButton, "underline")) {

        check = GTK _ CHECK _ MENU _ ITEM(menuUnderline);
    }

    /* --- If we found a button --- */
    if (check) {

        /* --- Set the state --- */
        gtk _ check _ menu _ item _ set _ state (check, nState);
    }
}
```

同样,如果菜单被选择,它也需要设置工具条按钮的状态。差别在于需要用 gtk _ toggle _ button _ set _ state 函数设置按钮的状态。而且你需要将构件转换为 GTK _ CHECK _ MENU _ ITEM 来确定按钮是否被选择。

```
void SelectMenu(GtkWidget *widget,gpointer data)
{
    GtkToggleButton *button = NULL;
    char *szButton;

    /* --- Get state of the menu --- */
    int nState = GTK_CHECK_MENU_ITEM(widget)->active;

    /* --- Show param and button state --- */
    szButton = (char *) data;
    printf("select menu %s - %d\n", szButton, nState);

    /* --- Toggle the toolbar button --- */
    if (!strcmp(szButton, "bold")) {

        button = GTK_TOGGLE_BUTTON(tool_bold);
    } else if (!strcmp(szButton, "italics")) {

        button = GTK_TOGGLE_BUTTON(tool_italics);
    } else if (!strcmp(szButton, "underline")) {

        button = GTK_TOGGLE_BUTTON(tool_underline);
    }
    if (button) {
        gtk_toggle_button_set_state(button, nState);
    }
}
```

这两个子程序非常相似。现在，当选择了工具条字体格式按钮，将对菜单选项的状态进行修改。当选择了菜单选项，工具条按钮的状态改变为相应的状态。这一小小的改动使应用程序更切合实际。没有任何事情会比工具条按钮和菜单选项显示不同状态更为糟糕。也许还有糟糕的事情，但是可以肯定对应用程序的用户来说，它的效果并不好。

5.12　小结

应用程序需要有良好的用户界面。菜单、工具条、像元图和工具提示有助于建立具有专业外观的应用。建立菜单项有两种方法：困难的方法和容易的方法。困难的方法更为灵活，也使你能做更多的事情，如在菜单中增加检查框等。可以用许多函数将构件加到工具条中。像元图是可以插入像元图构件的影像，它也可以插入按钮以提供带有图标的按钮。在本章中建立了小小的应用程序，用来说明如何用手工编程和使用套件来建立 shell 应用程序。

第6章　更多的构件

简单构件只是使用 GTK + 的开始。GTK + 有许多用来帮助开发人员编写复杂应用的构件。本章包含部分附加的构件，其中包括框架构件、对话框构件、文件选择对话框构件、进度条和强有力的正文构件。

6.1　框架

框架是用来组合构件的容器。通常它有标题和围绕周围的边框。框架有助于在屏幕上区分不同的区域。使用 gtk _ frame _ new 函数建立框架，它建立默认的框架。建立框架的标准过程如下：

```
/* ---Create the frame--- */
frame = gtk_frame_new("Title");

/* ---Set any attributes--- */

/* ---Make it visible--- */
gtk_widget_show(frame);

/* ---Add it to a container or a box or something--- */
gtk_container_add(window, frame);
```

建立框架以后，可以使用 gtk _ frame _ set _ label 函数改变框架的标号。

```
gtk_frame_set_label(frame, "Hey, new title");
```

可以使用 gtk _ frame _ set _ shadow 函数设置边框的特征。它确定框架的外观。用 GTK _ SHADOW _ IN 设置框架使整个框架看上去下陷。用 GTK _ SHADOW _ OUT 设置框架使框架的外观高于其他构件。这种效果是利用眼睛的幻觉达到的。最常用的是 GTK _ SHADOW _ ETCHED _ OUT 和 GTK _ SHADOW _ ETCHED _ IN，只画框架的边框。而 GTK _ SHADOW _ ETCHED _ NONE 则不画边框。图 6.1 表示不同类型的框架。

在框架中，标题可以出现在 3 个不同的位置：左上角、居中或右上角。可以用函数 gtk _ frame _ set _ label _ align 改变对齐方式。用于设置标号对齐的参数同样适用于设置框架标号的对齐。0 表示左对齐，.5 表示居中，1 表示右对齐。可以从下面的例子中看到这些参数的使用方式：

```
/* ---Set frame label to the left--- */
gtk_frame_set_label_align(GTK_FRAME(frame), 0, 0);
```

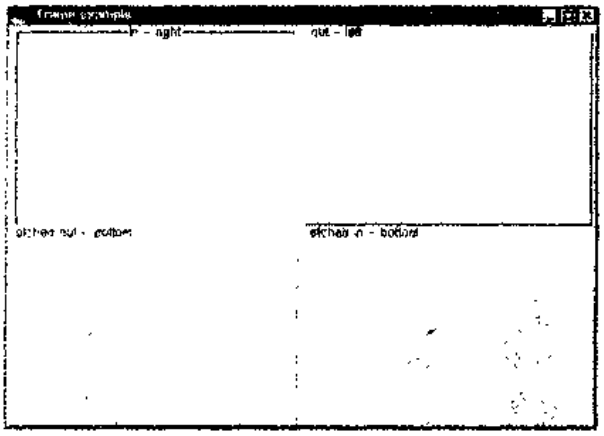


图 6.1　框架

```
/* --- Center the frame label --- */
gtk_frame_set_label_align (GTK_FRAME (frame), .5, 0);

/* --- Put the frame label on the right --- */
gtk_frame_set_label_align (GTK_FRAME (frame), 1, 0);
```

当建立框架后,可以将构件加入框架。应该将框架作为带有标号和边框的容器来对待。如果需要将一个以上的构件加到框架中,同样需要用到组合框或组合表构件。

6.2　正文构件

正文构件(见图 6.2)类似录入构件(见第 4 章"基本构件")。但是,正文构件支持多行正文,使它适合作为编辑程序。事实上,在后面建立编辑程序的例子中就用到了正文构件。正文构件远比较小的构件复杂,可以对它设置许多特征。

正文构件支持一系列快捷键,包括标准的剪切、复制和粘贴的快捷键。录入构件同样支持这些快捷键,因为它们都从 GtkEditable 派生而来。下面是正文构件支持的一些快捷键。

移动快捷键:

- Ctrl + A ——行的开始
- Ctrl + E ——行的末尾
- Ctrl + N ——下一行
- Ctrl + P ——前一行
- Ctrl + B ——后移一个字符
- Ctrl + F ——前移一个字符
- Alt + B ——后移一个字
- Alt + F ——前移一个字

编辑快捷键:

- Ctrl + H ——删除后一个字符(backspace)
- Ctrl + D ——删除前一个字符(delete)
- Ctrl + W——删除后一个字
- Alt + D ——删除前一个字
- Ctrl + K ——删到行的末尾
- Ctrl + U ——删除行

选择快捷键:

- Ctrl + X ——剪切到剪切板
- Ctrl + C ——复制到剪切板
- Ctrl + V ——从剪切板粘贴

正文构件几乎就是一个编辑程序。使用正文构件建立编辑程序不需要做多少事情。在第8章"开发简单的字处理程序"中,你将看到用正文构件建立小型编辑程序的过程是非常简单的。

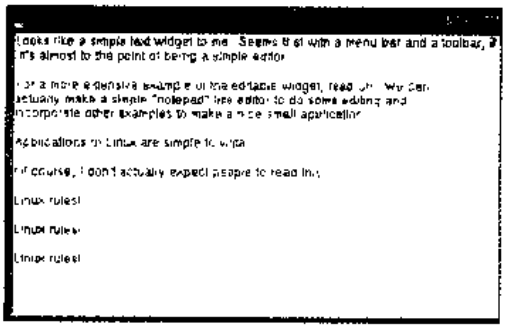


图 6.2 正文构件

6.2.1 建立正文构件

使用 gtk _ text _ new 函数建立正文构件,同时传递 GtkAdjustment 值作为纵向和横向调整值。通常传递 NULL 作为缺省调整参数。可以建立正文构件如下:

```
text = gtk_text_new (NULL, NULL);
```

可以使用 gtk _ text _ set _ editable 函数将正文构件设置为只读。如果传递的参数为 FALSE,将正文构件设置为只读。否则,正文构件是可以编辑的。缺省设置是可以编辑的。

```
/* --- Allow the text widget to be edited --- */
gtk_text_set_editable (GTK_TEXT (text), TRUE);

/* --- Set the widget to "read-only" mode --- */
gtk_text_set_editable (GTK_TEXT (text), FALSE);
```

光标位置可以用 gtk _ text _ set _ point 函数进行设置。需要为这一函数提供确定光标位置的索引值(index)。索引值是基于字符的索引。例如:

```
/* --- Set the cursor after the tenth character --- */
gtk_text_set_point (GTK_TEXT (text), 10);
```

可以使用 gtk_text_get_point 函数取得当前的光标位置。调用这一函数将返回索引值，它表示光标在正文中的位移。

```
/* --- Find out where the cursor is --- */
npos = gtk_text_get_point (GTK_TEXT (text));
```

可以使用 gtk_text_get_lenth 函数确定在正文构件中有多少个字符，还可以用 gtk_editable_get_chars 函数取得正文构件中的正文。例如，要取得正文构件中的所有字符，可以使用下面的代码：

```
/* --- Get the number of chars in the widget --- */
nLength = gtk_text_get_lenth (GTK_TEXT (text));

/* --- Get the text in the widget --- */
buffer = gtk_editable_get_chars (GTK_EDITABLE (text),
                                 (gint) 0,
                                 (gint) nLength);
```

可以用 -1 值表示最后的字符以代替计算构件中正文的长度：

```
buffer = gtk_editable_get_chars (GTK_EDITABLE (text),
                                 (gint) 0,
                                 (gint) -1);
```

在函数 gtk_editable_get_chars 中返回的缓冲区(buffer)是专门为返回值分配的新的内存块。在退出应用以前让这样的内存块留在内存不是好的程序设计实践。(你不这样认为？为什么？你认为这是 Macrosoft 的应用？)当这样的内存块不再需要时，应该用 g_free 函数释放它。

```
/* --- Done with the memory, free it --- */
g_free (buffer);
```

可以使用 GTK_TEXT_INDEX 宏指令取得正文构件中的单个字符。给宏指令传递索引值，将取得这一位置的字符。

```
/* --- Get the 101st character in the widget --- */
ch = GTK_TEXT_INDEX (GTK_TEXT (text), 101);
```

可以使用 gtk_text_insert_text 函数将正文插入构件。这一函数在当前的光标位置插入正文。如果增加的正文使用特定的字体和颜色，可以使用带有字体和颜色信息的 gtk_text_insert 函数。传递 NULL 参数将使用当前的字体和颜色。

```
/* --- Add some text without a font/color --- */
```

```
gtk_text_insert (GTK_TEXT (text), NULL, NULL, NULL,
                 buffer, strlen (buffer));

/* --- Add some text with a font/color --- */
gtk_text_insert (GTK_TEXT (text), font,
                 foreColor, backColor,
                 buffer, strlen (buffer));

/* --- Add some text and use current font/color --- */
gtk_text_insert_text (GTK_TEXT (text), buffer,
                      strlen (buffer), &nPosition);
```

相对于当前的光标位置删除正文。可以从当前的光标位置向前或向后删除正文。函数 gtk_text_forward_delete 从当前光标位置向前删除字符。函数 gtk_text_backward_delete 从当前光标位置向后删除字符。

```
/* --- Delete the ten character after the cursor --- */
bSuccess = gtk_text_forward_delete (GTK_TEXT (text), (guint) 10);

/* --- Delete the ten character before the cursor --- */
bSuccess = gtk_text_backward_delete (GTK_TEXT (text), (guint) 10);
```

如果成功地进行了删除,返回 TRUE;否则,返回 FALSE。

6.2.2 插入和删除正文

当需要删除和插入大量的正文时,删除和插入正文的操作可能会很慢。在每次删除或插入正文以后,构件本身需要进行重新绘制。特别是当需要删除和插入若干字符串时,这不是好的做法。有时,好的做法是在删除或插入正文以前先挂起构件的更新操作。你可以调用 gtk_text_freeze 函数挂起构件的自动的更新操作,并调用 gtk_text_thaw 函数重新开始更新操作。在更新构件以前调用 gtk_text_freeze 函数。当完成更新操作后,调用 gtk_text_thaw 函数结束挂起。如果在挂起期间进行了更新,这将导致构件进行重新绘制。可以使用类似下面的代码来实现上面提到的过程:

```
/* --- Freeze the widget --- */
gtk_text_freeze (GTK_TEXT (text));

/* --- Put the updates here --- */

/* --- Un-freeze the widget --- */
gtk_text_thaw (GTK_TEXT (text));
```

如果在大量插入和删除正文时不使用 gtk_text_freeze 函数和 gtk_text_thaw 函数,将导致构件在每次删除或插入以后进行重新绘制。代替 [insert text(插入正文)] [insert text(插

入正文)] | insert text(插入正文)] [paint(绘制)],构件将实际执行[insert text(插入正文)] [paint(绘制)] [insert text(插入正文)] [paint(绘制)] [insert text(插入正文)][paint(绘制)]。两个附加的绘制是不必要的,因为有附加的操作需要完成。好的做法是告诉构件直到所有的删除和插入操作完成后,才对它自身进行更新。如果你使用编辑程序实现搜索和替换就可能发生这样的情况。最好在进行更新操作时"冻结"屏幕,并只显示最后的结果。而不是当每次在构件内加入新的正文时,构件就进行重新绘制因而减慢了搜索和替换。

正文构件是某种类型的编辑程序,它必须支持剪切、复制和粘贴操作。在正文构件内自动支持剪切(Ctrl + X)、复制(Ctrl + C)和粘贴(Ctrl + V)。然而,许多人倾向于使用鼠标来处理诸如复制、剪切和粘贴等功能。你可以在构件内调用处理复制、剪切和粘贴功能的函数来实现这些功能。

函数 gtk _ editable _ copy _ clipboard 从构件复制选择的正文,并将它移到剪切板　复制的正文继续保存在剪切板上直到被覆盖为止。

```
/* --  Copy selected text and move to the clipboard --- */
gtk_editable_copy_clipboard(GTK_EDITABL(text), GDK_CURRENT_TIME);
```

函数 gtk _ editable _ cut _ clipboard 从构件移去选择的正文,并将它移到剪切板。

```
/* --- Remove the selected text from the widget and put in the clipboard --- */
gtk_editable_cut_clipboard(GTK_EDITABL(text), GDK_CURRENT_TIME);
```

函数 gtk _ editable _ paste _ clipboard 从剪切板粘贴正文,并将它插入构件内当前光标所在的位置。

```
/* --- Copy the text from the clipboard and insert it into --- */
/* --  the widget at the cursor position --- */
gtk_editable_paste_clipboard(GTK_EDITABL(text), GDK_CURRENT_TIME);
```

6.2.3　滚动条

滚动条(Scrollbar)在正文构件内很有用。例如,当用户送入构件的正文超过屏幕显示的范围时它就有用。这种情况经常发生。如果没有滚动条,虽然将正文放在构件内,用户用鼠标浏览正文仍然是困难的。有了滚动条,允许用户用鼠标滚动正文,正文构件就更有用。

建立滚动条、并将它和正文控制结合在一起是容易的。下面的代码在组合表内建立正文构件,并增加用于编辑控制的滚动条。建立 2×2 的组合表,以便将正文构件放在其中;在正文构件右侧放纵向滚动条,在正文构件下面放横向滚动条。在组合表内,将正文构件放在组合表的左上角(0-1,0-1);将纵向滚动条放入组合表的右上角(1-2,0-1);将横向滚动条放入组合表的左下角(0-1,1-2)。

```
/*    - Create the widget --- */
table = gtk_table_new(2, 2, FALSE);

/* --- Make it visible --- */
```

```
gtk_widget_show (table);
/* --- Create the widget --- */
text = gtk_text_new (NULL, NULL);
/* --- Add the widget to the packing table --- */
gtk_table_attach (GTK_TABLE (table), text, 0, 1, 0, 1,
        GTK_EXPAND | GTK_SHRINK | GTK_FILL,
        GTK_EXPAND | GTK_SHRINK | GTK_FILL, 0, 0);

/* --- Make it visible --- */
gtk_widget_show (text);
```

当将正文构件放入组合表后,可以将滚动条加到构件中去。必须将滚动条加到正文构件的结构中。这一结构有 hadj 字段和 vadj 字段,这些是 GtkAdjustment 指针。建立滚动条,并将正文字段的 GtkAdjustment 和滚动条联系在一起;这只要在建立滚动条时给它传递 GtkAdjustment 指针就可以了。

```
/* ------------------------------- */
/* --- Create a horizontal scrollbar --- */
/* ------------------------------- */
hscroll = gtk_hscrollbar_new (GTK_TEXT (text)->hadj);

/* --- Put the horizontal scrollbar beneath the text widget --- */
gtk_table_attach (GTK_TABLE (table), hscroll, 0, 1, 1, 2,
        GTK_EXPAND | GTK_FILL | GTK_SHRINK, GTK_FILL, 0, 0);

/* --- Display the widget --- */
gtk_widget_show (hscroll)

/* ----------------------------- */
/* --- Create a vertical scrollbar --- */
/* ----------------------------- */
vscroll = gtk_vscrollbar_new (GTK_TEXT (text)->vadj);

/* --- Put the vertical scrollbar to the right of the text --- */
gtk_table_attach (GTK_TABLE (table), vscroll, 1, 2, 0, 1,
        GTK_FILL, GTK_EXPAND | GTK_SHRINK | GTK_FILL, 0, 0);

/* --- Make it visible --- */
gtk_widget_show (vscroll);
```

如果不需要横向滚动条,建立组合表的工作可以简化,只要将纵向滚动条放在正文构件的右侧就可以了。下列代码是一个例子:

```
/* --- Create the widget --- */
text = gtk_text_new (NULL, NULL);

/* --- Add the text widget to the packing box --- */
gtk_box_pack_start (GTK_BOX (hbox), text, FALSE, FALSE, 0);

/* --- Create a vertical scrollbar --- */
vscroll = gtk_vscrollbar_new (GTK_TEXT(text)->vadj);

/* --- Add scrollbar.  Should be to the right of text widget --- */
gtk_box_pack_start(GTK_BOX(hbox), vscroll, FALSE, FALSE, 0);

/* --- Make the scrollbar visible --- */
gtk_widget_show (vscroll);
```

6.3　对话框

对话框(dialog box)可以被用来向用户显示信息(如“你的磁盘快满了”),或者向用户提问(如“你肯定需要做这件事?”),或者从用户那里取得信息(如“你需要将文件保存为什么类型?”)。

使用 gtk_dialog_new 函数建立对话框。它建立不带正文和按钮的基本对话框。建立的对话框有两个组合框(vbox,action_area),用来将其他构件放入其中。vbox 用来放标号或其他信息,将按钮放入 action_area。空白的对话框毫无用处,直到加入正文构件、按钮,并向用户传递信息它才成为有用。使对话框成为有用的最简单方法是显示一些正文,并带一个“Ok”按钮。可以用下面的代码建立按钮,并把它加到对话框的 vbox 组合框中。

```
/* --- Create the label --- */
label = gtk_label_new (szMessage);

/* --- Give it a little breathing room --- */
gtk_misc_set_padding (GTK_MISC (label), 10, 10);

/* --- Add it to the dialog box's vbox area --- */
gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog_window)->vbox),
                    label, TRUE, TRUE, 0);

/* --- Always make it visible --- */
gtk_widget_show (label);
```

现在它仍然没有提供任何与用户交流的信息。为做到这一点,应该将一个按钮加到 action_area。当你增加按钮到对话框时,最好对按钮设置 GTK_CAN_DEFAULT 标志。GTK

_CAN_DEFAULT 标志为对话框指定可以接受默认操作的构件,并允许用户按 Enter 键代替点击按钮。函数 gtk_widget_grab_default 将指定的构件设置为默认操作的构件。在这个简单的例子中只有一个按钮,因此将它作为对话框的默认按钮;当弹出对话框时,用户可以按 Enter 键。默认按钮带有很宽的边框以表示这是默认构件。增加"Ok"按钮的代码如下:

```
/* --- Create the button --- */
button = gtk_button_new_with_label ("Ok");

/* --- Allow the button to be defaulted --- */
GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);

/* --- Add widget to the action area --- */
gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog_window)->action_area),
                    button, TRUE, TRUE, 0);

/* --- Grab the focus --- */
gtk_widget_grab_default (button);

/* --- Make it visible --- */
gtk_widget_show (button);

/* --- Wait for the button to be clicked --- */
gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (OkClicked),
                    &return_value);
```

Linux 支持若干种类型的对话框。一些是独占模式(modal)对话框,另一些是无模式的对话框。独占模式对话框将应用的重点保留在对话框上,并使对话框一直留在那里;在对话框消失以前,用户不能使用任何其他窗口。唯一允许应用程序进行处理的事件是独占模式对话框。在 Linux 中,当需要确保用户看过信息,或者在对话框的系列操作中,这些信息对进入下一步是重要的,在这些情况下就显示独占模式对话框。在许多情况下,文件选择对话框是独占模式对话框,因为用户必须确定他或她要装入的文件。在用户选择文件和按"Ok"按钮以前做任何其他事情都是没有意义的,或者用户决定按"Cancel" 按钮撤消对话框。图 6.3 表示对话框的一些例子。

无模式对话框没有严格的依赖关系。在对话框中可能显示颜色图给用户提供颜色的选择。在这种情况下,可以同时显示其他窗口和对话框,不必将重点限制在颜色对话框。

在这里建立的 Popup 函数显示简单的对话框,并等待用户点击"Ok"按钮。整个过程包括:建立对话框,建立用于消息的标号,将标号置于对话框中,建立"Ok"按钮并将它置于对话框中。"Ok"按钮具有事件处理程序。当按下按钮时,它关闭对话框。这一对话框是独占模式对话框,因为它调用了 gtk_grab_add 函数。此函数确保只有这个对话框能得到事件信号。

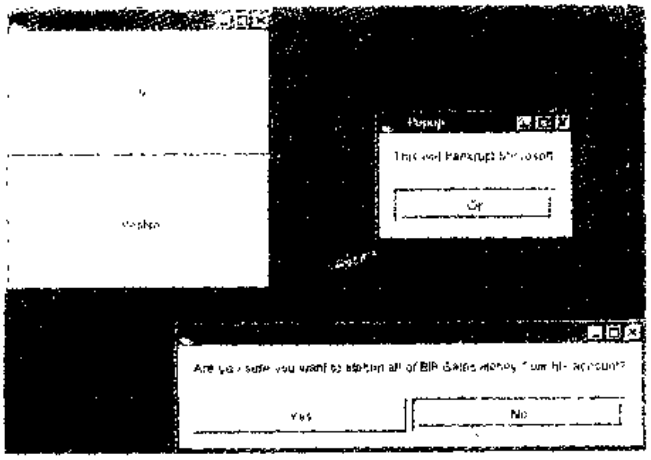


图 6.3　对话构件

```
/*
 * Popup
 *
 * Display a popup dialog window with a message and an
 * "Ok" button
 */
void Popup (char * szMessage)
{
    static GtkWidget * label;
    GtkWidget *button;
    GtkWidget *dialog_window;

    /* --- Create a dialog window --- */
    dialog_window = gtk_dialog_new ();

    /* --- Trap the window close signal to release the grab --- */
    gtk_signal_connect (GTK_OBJECT (dialog_window), "destroy",
                        GTK_SIGNAL_FUNC (ClosingDialog),
                        &dialog_window);

    /* --- Add a title to the window --- */
    gtk_window_set_title (GTK_WINDOW (dialog_window), "Popup");

    /* --- Create a small border --- */
    gtk_container_border_width (GTK_CONTAINER (dialog_window), 5);

    /* ------------------------ */
    /* --- Create the message --- */
```

```
/* ----  ------------------ */

/* ---Create the message in a label--- */
label = gtk_label_new (szMessage);

/* ---Put some room around the label--- */
gtk_misc_set_padding (GTK_MISC (label), 10, 10);

/* ---Add the label to the dialog box--- */
gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog_window)->vbox),
                    label, TRUE, TRUE, 0);

/* ---Make the label visible--- */
gtk_widget_show (label);

/* ------------------ */
/* ---"Ok" button--- */
/* ------------------ */

/* ---Create the "Ok" button--- */
button = gtk_button_new_with_label ("Ok");

/* ---Need to close the window if they press "Ok"--- */
gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (CloseDialog),
                    dialog_window);

/* ---Allow it to be the default button--- */
GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);

/* ---Add the button to the dialog box--- */
gtk_box_pack_start (GTK_BOX (
                GTK_DIALOG (dialog_window)->action_area),
               button, TRUE, TRUE, 0);

/* ---Make the button the default button--- */
gtk_widget_grab_default (button);

/* ---Make the button visible--- */
gtk_widget_show (button);

/* ---Make the dialog box visible--- */
```

```
    gtk_widget_show (dialog_window);

    gtk_grab_add (dialog_window);
}
```

映射到"Ok"按钮的 CloseDialog 调用 gtk_grab_remove 函数释放对话框的独占状态，然后关闭对话窗口。将对话框的 GtkWidget 指针作为 data 字段传递给函数，因为回呼函数并不知道信号来自哪一个构件。函数 gtk_signal_connect 建立传递对话窗口的信号到回呼函数的关系。在这里，回呼函数用来关闭对话窗口。

```
/*
 * CloseDialog
 *
 * Close the dialog window.  The dialog handle is passed
 * in as the data.
 */
void CloseDialog (GtkWidget *widget, gpointer data)
{

    /* --- Close it --- */
    gtk_widget_destroy (GTK_WIDGET (data));

}
```

当建立对话窗口时，调用了 gtk_grab_add 函数，使对话窗口处于独占模式。在关闭对话窗口时，相应需要调用 gtk_grab_remove 函数释放对话框的独占状态。"destroy"信号正好用来做这件事。因为它表示正在关闭(closing)窗口。

```
/*
 * ClosingDialog
 *
 * Calls when window is about to close.  Returns FALSE
 * to let it close.  Releases the grab which made the dialog
 * window modal.
 */
void ClosingDialog (GtkWidget *widget, gpointer data)
{
    gtk_grab_remove (GTK_WIDGET (widget));
}
```

稍复杂的例子是"Yes/No"或"Ok/Cancel"类型的对话框。它们向用户显示信息，并提供选择。通常它们带有类似"Are you sure you want to do this?"的信息，在对话框中作出的选择将导致不同事情的发生。你能够扩充前面的例子来建立这种类型的对话框，给对话框的每个

按钮设置其自己的事件处理程序。

注意:在下面的例子中,对"Yes"和"No"按钮都设置了 GTK_CAN_DEFAULT 标志。每一个按钮都能为对话框实现默认的操作。如果选择其中一个按钮,并按下鼠标按钮,围绕按钮外围的宽边框将移到这一按钮上。此时按 Enter 键相当于按默认的按钮。使用 gtk_widget_grab_default 函数将"No"按钮作为对话框开始使用的默认按钮。你必须经常将非破坏性操作作为默认操作。例如,当一个重新格式化硬盘驱动器的程序提示"Are you sure you want to do this(你确实需要这样做吗?)",应该将"No"按钮作为默认操作。

函数 YesNo 提供消息("Are you sure you want to do this ?")和两个函数。当用户按下"Yes"按钮时调用 YesFunc 函数;当用户按下"No"按钮时调用 NoFunc 函数。

```
/*
 * YesNo
 *
 * Function to display a yes/no window
 */
void YesNo (char * szMessage, void (* YesFunc)(), void (* NoFunc)())
{
    GtkWidget *label;
    GtkWidget *button;
    GtkWidget *dialog_window;

    /* --- Create the dialog box --- */
    dialog_window = gtk_dialog_new ();

    /* --- Trap the window close signal to release the grab --- */
    gtk_signal_connect (GTK_OBJECT (dialog_window), "destroy",
                        GTK_SIGNAL_FUNC (ClosingDialog),
                        &dialog_window);

    /* --- Set the title --- */
    gtk_window_set_title (GTK_WINDOW (dialog_window), "YesNo");

    /* --- Add a small border --- */
    gtk_container_border_width (GTK_CONTAINER (dialog_window), 5);

    /* ------------------------ */
    /* --- Create the message --- */
    /* ------------------------ */

    /* --- Create a label with the message --- */
    label = gtk_label_new (szMessage);
```

```
/* --- Give the label some room --- */
gtk_misc_set_padding (GTK_MISC (label), 10, 10);

/* --- Add label to the dialog box --- */
gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog_window)->vbox),
                    label, TRUE, TRUE, 0);

/* --- Show the widget --- */
gtk_widget_show (label);

/* ---------------- */
/* --- Yes button --- */
/* ---------------- */

/* --- Create the "Yes" button --- */
button = gtk_button_new_with_label ("Yes");

gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (YesFunc),
                    dialog_window);

GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);

/* --- Add the button to the dialog box --- */
-gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog_window)? ->action  area),
     button, TRUE, TRUE, 0);

/* --- Make the button visible --- */
gtk_widget_show (button);

/* ---------------- */
/* --- No button --- */
/* ---------------- */

/* --- Create the "No" button --- */
button = gtk_button_new_with_label ("No");

gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (NoFunc),
                    dialog_window);

/* --- Allow "No" to be a default --- */
```

```
    GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);

    /* --- Add the "No" button to the dialog box --- */
    gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog_window)->action_area),
        button, TRUE, TRUE, 0);

    /* --- Make the "No" the default --- */
    gtk_widget_grab_default (button);

    /* --- Make the button visible --- */
    gtk_widget_show (button);

    /* --- Show the dialog box --- */
    gtk_widget_show (dialog_window);

    /* --- Only this window can be used for now --- */
    gtk_grab_add (dialog_window);
}
```

现在应用程序可以使用 YesNo 函数建立极妙的对话框。可以按如下方式调用函数：

```
YesNo ("Are you sure you want to siphon all of"
       "Bill Gates money from his account?",
       DisplayYes, DisplayNo);
```

当函数 DisplayYes 和 DisplayNo 完成处理后，应该撤消对话框。函数 YesNo 传递对话框的 GtkWidget 参数作为 clicked 回呼函数中的 gpointer。这里使用的 DisplayYes 和 DisplayNo 函数很简单，当需要它们进行更多的处理时就会变得更长。

```
/*
 * DisplayYes
 *
 * Show that the Yes button was pressed
 */
void DisplayYes (GtkWidget *widget, gpointer data)
{
    /* --- Display message --- */
    g_print ("Withdrawing $ 60,000,000,000.00\n");

    /* --- Close it --- */
    gtk_widget_destroy (GTK_WIDGET (data));
}
```

```
/*
 * DisplayNo
 *
 * Show that the No button was pressed
 */
void DisplayNo (GtkWidget *widget, gpointer data)
{
    /* --- Display message --- */
    g_print ("Money can't buy you happiness. \n");

    /* --- Close it --- */
    gtk_widget_destroy (GTK_WIDGET (data));

}
```

6.4　文件选择对话框

文件选择对话框是编好程序的用来提供标准“文件选择”的专用对话框，它类似 Windows 所具有的通用“文件选择”。这一特征使它具有 GTK + 应用固有的外观和感受。文件选择对话框允许用户选择文件，并对文件完成一些操作(见图 6.4)。

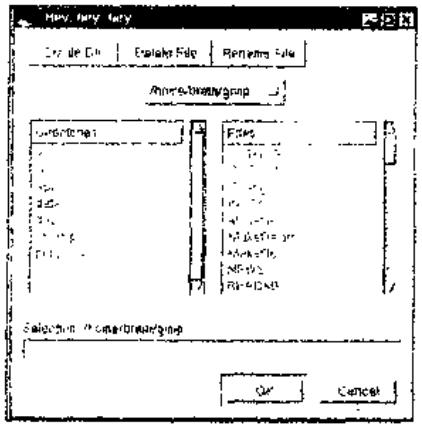


图 6.4　文件选择对话框

在 GTK + 中，使用 gtk_file_selection_new 函数建立文件选择对话框。需要给函数传递用于对话窗口的标题。当用户在对话框出现后忘了他或她正在做什么或者被中断时，将标题设置为类似 Save as 和 Open 可能是好的做法。

```
/* --- Create the dialog box with a title --- */
filew = gtk_file_selection_new ("Save as");
```

如果由于用户选择了"Save as"菜单后显示对话框,应该将对话框中的文件名填为当前的文件名(也许还有其他理由需要这样做)。使用 gtk _ file _ selection _ set _ filename 函数,可以使对话框用特定的文件名作为默认的文件名。

```
/* ... Set the default filename ... */
gtk_file_selection_set_filename (
                    GTK_FILE_SELECTION(filew),
                    "problems.txt");
```

文件选择对话框具有一系列开发人员可以访问的构件。在 gtkfilesel.h 文件中定义的 GtkFileSelection 结构如下:

```
struct _GtkFileSelection
{
  GtkWindow window;

  GtkWidget *dir_list;
  GtkWidget *file_list;
  GtkWidget *selection_entry;
  GtkWidget *selection_text;
  GtkWidget *main_vbox;
  GtkWidget *ok_button;
  GtkWidget *cancel_button;
  GtkWidget *help_button;
  GtkWidget *history_pulldown;
  GtkWidget *history_menu;
  GList     *history_list;
  GtkWidget *fileop_dialog;
  GtkWidget *fileop_entry;
  gchar     *fileop_file;
  gpointer   cmpl_state;

  GtkWidget *fileop_c_dir;
  GtkWidget *fileop_del_file;
  GtkWidget *fileop_ren_file;

  GtkWidget *button_area;
  GtkWidget *action_area;

};
```

常用的构件有:ok _ button、cancel _ button 和 help _ button。对大多数其他构件定义了默认的操作,通常不需要对它们加以改变。按钮是文件选择对话框的一部分,可以用它们登记你

自己的事件。例如，如果你需要知道什么时候按了“Cancel”按钮，可以用一小段代码对 GtkFileSelection 结构中的 cancel_button 设置事件处理程序。

```
gtk_signal_connect_object (
        GTK_OBJECT (GTK_FILE_SELECTION (filew)->cancel_button),
        "clicked",
        (GtkSignalFunc) gtk_widget_destroy,
        GTK_OBJECT (filew));
```

函数 gtk_signal_connect_object 和函数 gtk_signal_connect 类似，它们都被用来设置信号的回呼函数。函数 gtk_signal_connect_object 为信号设置回呼函数；当它调用回呼函数时，它只用函数 gtk_signal_connect_object 的第 4 个参数。这相当于使回呼函数不了解导致调用回呼函数的构件。在这种情况下的回呼函数具有类似如下的形式：

```
gint CallbackFunction (GtkObject *object);
{
}
```

传递给回呼函数的对象(object)是在函数 gtk_signal_connect_object 中设置的文件选择构件(filew)。任何引用导致事件出现的构件(本例中的“Cancel”按钮)被移去。但在这种情况下，这一点并不重要。

使用类似的事件处理程序，可以设置“Ok”按钮对文件完成某些操作；如果要使应用程序变为对用户友好，可以配置“Help”按钮完成某些工作。(也可以使它不出现，因而用户事先并没有形成取得帮助的概念；但这样做就不是用户友好的应用)

6.5　进度条

进度条(progress bar，GtkProgressBar)是极好的小构件。通常用它显示耗费时间的函数的当前状态。显示一项操作的进度要比用户不知道该项操作要化多长时间才能完成好。开发人员对进度条构件具有全面的控制能力，因为他或她需要写程序去更新构件使它正确地显示进度。GtkProgressBar 从 GtkProgress 构件继承许多功能。它们的许多功能不必在这里加以讨论。我们将建立用于本书其余部分的基本的进度条，当我们在完成耗费时间的操作时就用它来显示进度(见图 6.5)。

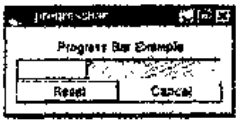


图 6.5　进度条

用 gtk_progress_bar_new 函数或者 gtk_progress_bar_new_with_adjustment 函数建立进度条构件。在建立后，可以用 gtk_progress_set_percent 函数更改进度条构件的显示，并

传递新的完成值作为参数。其中 0 表示完全没有完成,1 表示全部完成。

这一构件使用数据结构保持对进度条信息的跟踪。信息包括:对话构件,进度条构件,指示对话框是否能关闭的标志,以及最新已知的在进度条构件上显示百分比。

```
typedef struct {

    GtkWidget *progressbar;
    GtkWidget *window;
    int bProgressUp;
    int nLastPct;

} typProgressData;
```

我们的进度条将用到 3 个函数。第一个用到的是 StartProgress 函数。它在对话框内建立进度条。

```
/*
 * StartProgress
 *
 * Create a window for the progress bar
 */
void StartProgress ()
{
    GtkWidget *label;
    GtkWidget *table;
    GtkWidget *window;
    GtkAdjustment *adj;

    pdata = g_malloc (sizeof (typProgressData));
    pdata->nLastPct = -1;
    pdata->bProgressUp = TRUE;
    /*
     * --- Create the top-level window
     */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    pdata->window = window;

    /* --- Hook up the destroy --- */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (CanWindowClose), pdata);

    gtk_container_border_width (GTK_CONTAINER (window), 10);
```

```
    /* --- Create a table --- */
    table = gtk_table_new (3, 2, TRUE);
    gtk_container_add (GTK_CONTAINER (window), table);

    /* --- Add a label to the table --- */
    label = gtk_label_new ("Loading...");
    gtk_table_attach_defaults (GTK_TABLE (table), label, 0,2,0,1);
    gtk_widget_show (label);

    /* --- Add the progress bar to the table --- */
    adj = (GtkAdjustment *) gtk_adjustment_new (0, 0, 400, 0, 0, 0);
    pdata->progressbar = gtk_progress_bar_new_with_adjustment (adj);
    gtk_table_attach_defaults (GTK_TABLE (table),
                               pdata->progressbar, 0,2,1,2);
    gtk_widget_show (pdata->progressbar);

    /*  -- Show everything --- */
    gtk_widget_show (table);
    gtk_widget_show (window);
}
```

函数 UpdateProgress 用耗费时间的操作的当前状态更改进度条。它取两个值——当前状态值和最终状态值。当当前状态值和最终状态值相等时，完成全部操作。在读入大的文件时可能用到进度条。最终状态值是文件的容量，而当前状态值则是读入文件的位置。

```
/*
 * UpdateProgress
 *
 * Update the progress window to reflect the state
 * of the file that is being loaded.
 *
 * pos - how much of the file has been loaded.
 * len - length of the file
 * (pos / len) = % file has been loaded.
 */
void UpdateProgress (long pos, long len)
{
    gfloat pvalue;
    int pct;

    /* --- Prevent divide by zero errors --- */
    if (len > 0) {
```

```
        /* ---Calculate the percentage--- */
        pvalue = (gfloat) pos / (gfloat) len;

        pct = pvalue * 100;

        if (pdata->nLastPct != pct) {

            /*  --Update the displayed value--- */
            gtk_progress_set_percentage (GTK_PROGRESS (pdata->progressbar),
                                         pvalue);

            /* ---Repaint any windows - like the progress bar--- */
            while (gtk_events_pending ()) {
            >gtk_main_iteration ();
            }
            pdata->nLastPct = pct;
        }
    }
}
```

函数 EndProgress 关闭带有进度条的对话框。在操作完成后调用这一函数关闭对话框。

```
/*
 * EndProgress
 *
 * Close down the progress bar.
 */
void EndProgress ()
{
    /* ---Allow it to close--- */
    pdata->bProgressUp = FALSE;

    /* ---Destroy the window--- */
    gtk_widget_destroy (pdata->window);

    /* ---Free used memory.--- */
    g_free (pdata);

    pdata = NULL;
}
```

函数 CanCloseWindow 阻止用户关闭窗口,一直到操作完成为止。当用户试图关闭对话窗口时,"delete_event(删除事件)"信号调用这一函数。返回值表示是否可以关闭窗口。

```
/*
 * CanWindowClose
 *
 * Function that determines that if the dialog
 * window can be closed.
 */
gint CanWindowClose (GtkWidget *widget)
{
    /* --- TRUE => cannot close --- */
    /* --- FALSE => can close --- */
    return (pdata->bProgressUp);
}
```

为了说明进度条，下面的例子建立窗口和窗口内的一个按钮。当按下按钮时，使用 StartProgress 函数建立带有进度条的对话框，并用定时器每隔 100ms 对进度条更改一次。

6.5.1　使用定时器

使用 gtk_timeout_add 函数建立定时器。这一函数取时间间隔、函数和 data 作为参数。将按照指定的时间间隔调用传递给 gtk_timeout_add 函数的函数，并将 data 参数传递给它作为参数。例如，下列的函数调用：

```
ptimer = gtk_timeout_add (100, UpdateProgressTimer, data);
```

每隔 100ms 调用 UpdateProgressTimer 函数一次，并用 data 作为它的参数。返回值是定时器的 ID，在结束定时器时需要用到它。使用函数 gtk_timeout_remove 来结束对上述函数的调用。函数 gtk_timeout_remove 使用的参数是 gtk_timeout_add 函数的返回值。

```
gtk_timeout_remove (ptimer);
```

6.5.2　测试进度条

用来测试进度条的完整程序如下：

```
/*
 * Application code begins here.
 */

#include <gtk/gtk.h>

int ptimer;
int nValue;

/*
```

```
 * UpdateProgressTimer
 *
 * This is the timer callback function.  It updates the
 * progress bar and closes it when it reaches the end.
 */
UpdateProgressTimer (gpointer data)
{

    /* --- Increment value --- */
    nValue += 1;

    /* --- Update the progress bar --- */
    UpdateProgress (nValue, 100);

    /* --- Close if it's at the end. --- */
    if (nValue == 100)
    {
        EndProgress ();
        gtk_timeout_remove (ptimer);
    }

}

/*
 * ButtonClicked
 *
 * Called to create the progress bar and the time when
 * the button has been clicked.
 */
gint ButtonClicked (GtkWidget *widget, gpointer data)
{
    /* --- Do some initialization --- */
    nValue = 0;
    StartProgress ();

    /* --- Call the timer. --- */
    ptimer = gtk_timeout_add (100, UpdateProgressTimer, data);
}


/*
 * CloseAppWindow
 *
 * Close down the application
```

```
 */
gint CloseAppWindow ()
{
    gtk_main_quit ();
    return (FALSE);
}

/*
 * main
 *
 * program begins here.
 */
main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *button;

    gtk_init (&argc, &argv);

    /* --- Create main window and give it a title --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Progress bar");

    /* --- End app when close window detected --- */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (CloseAppWindow), NULL);

    /* --- Create a button to bring up the progress bar --- */
    button = gtk_button_new_with_label ("Progress bar");
    gtk_widget_show (button);

    /* --- This is where we handle it --- */
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (ButtonClicked), NULL);

    gtk_container_add (GTK_CONTAINER (window), button);
    gtk_widget_show (window);

    /* --- Let GTK take over --- */
    gtk_main ();
    exit (0);
}
```

6.6 小结

在本章中讨论的构件比在前一章中说明的构件更为复杂。在 GTK + 中,广泛使用对话框。用文件选择构件说明如何使用对话框。文件选择对话框是允许用户选择文件的标准对话框。正文构件本身几乎是一个编辑程序。进度条有助于显示耗费时间的操作的进度。所有这些构件使得在 Linux 中编写应用程序变得简单。

第二部分　应用举例

第 7 章　编写计算器应用
第 8 章　开发简单的字处理程序
第 9 章　扫雷

第7章　编写计算器应用

在本章中，将运用已掌握的知识编写简单的计算器程序。这是一个很好的启蒙程序，因为它包含了一组按钮和可以显示结果的区域。将按钮和显示区一起整齐地放在组合表(PackingTable)中。应用程序接受按键或点击鼠标事件将用户输入的信息送入完成简单算术运算的计算器。

7.1　计算器程序

当用户在应用中完成操作时，给用户提供反馈信息是很重要的。如果用户用鼠标点击计算器的按钮，他或她应该看到按钮被按下的样子——也就是说，按钮应该有被按下并反弹回来的视觉效果。另外，显示区也应该根据点击的按钮进行刷新。当然，有些用户喜欢用键盘。当他们从键盘输入时，同样也应该得到反馈。当用户在键盘上按键时，应用程序应该将重点转移到对应的按钮。通过这种方式输入信息的用户将看到他或她按下的键被突出地显示，同时显示区也被更新。

7.1.1　数据结构

在计算器中的每个按钮都有一组各自的特征。每个按钮都有构件指针、标号和它的位置信息。其中位置信息以行、列格式表示，此信息决定了按钮在组合表中的位置。

每个按钮的数据结构如下：

```
/*
 * --- Data structure to keep track of the calculator buttons.
 */
typedef struct {

    char        *szLabel;     /* --- Label display on button --- */
    int         row;          /* --- Row to place the button --- */
    int         col;          /* --- Column to place the button --- */
    GtkWidget   *widget;      /* --- The button pointer --- */

} typCalculatorButton;
```

可以在组合表中用这个结构定义按钮。这个表是由 typCalculatorButton 元素组成的数组。在表中定义按钮的好处是你可以很容易地在表中添加或删除按钮而不用对原始代码进行太多的改动。事实上，你甚至可以在表中移动按钮而不必改变其他的代码行。按钮的定义如

下:

```
/*
 * ---This is the button list.  Each button is documented here so
 *      we can access it.
 */
typCalculatorButton buttonList [] = {
    {"C",    1, 0, NULL},      /* ---Clear--- */
    {"CE",   1, 1, NULL},      /* ---Clear--- */
    {"/",    1, 3, NULL},      /* ---Division--- */

    {"7",    2, 0, NULL},      /* ---Digit--- */
    {"8",    2, 1, NULL},      /* ---Digit--- */
    {"9",    2, 2, NULL},      /* ---Digit--- */
    {"*",    2, 3, NULL},      /* ---Multiplication--- */
    {"%",    2, 4, NULL},      /* ---Percent--- */

    {"4",    3, 0, NULL},      /* ---Digit--- */
    {"5",    3, 1, NULL},      /* ---Digit--- */
    {"6",    3, 2, NULL},      /* ---Digit--- */
    {"-",    3, 3, NULL},      /* ---Subtraction--- */
    {"1/x",  3, 4, NULL},      /* ---1/x--- */

    {"1",    4, 0, NULL},      /* ---Digit--- */
    {"2",    4, 1, NULL},      /* ---Digit--- */
    {"3",    4, 2, NULL},      /* ---Digit--- */
    {"+",    4, 3, NULL},      /* ---Addition--- */
    {"sqrt", 4, 4, NULL},      /* ---Square root--- */

    {"+/-",  5, 0, NULL},      /* ---Negate value--- */
    {"0",    5, 1, NULL},      /* ---zero--- */
    {".",    5, 2, NULL},      /* ---Decimal--- */
    {"=",    5, 3, NULL},      /* ---Equals/total--- */
    {"X2",   5, 4, NULL},      /* ---Square--- */
};
```

在表中,下面的公式用于计算按钮的个数。

```
int nButtons = sizeof(buttonList)/
               sizeof(typCalculatorButton);
```

这个公式在编译的时候自动计算出数组中按钮的个数,这样你在添加或删除按钮时不必手工调节按钮的个数值。这比给数组定义好固定长度、然后再在每次添加或删除按钮时改变

数组长度的方法要好得多。

除了定义按钮信息的表(buttonList)以外,应用程序还需要其他的全局变量。例如,用于显示结果的标号。这个标号就相当于计算器的 LCD 显示屏幕。在用户不能直接向构件输入的情况下,控制输入到 LCD 显示屏上的信息是比较容易的。应用窗口可以捕捉到任何按键操作。应用程序过滤掉不在显示区中的字符后刷新标号。

7.1.2　主程序

我们已经对计算器和它的数据结构进行了解释,现在你可以编写程序了。主函数完成启动计算器所需要的设置、创建窗口和定位,还要放置一些信号回呼函数。这样,就可以调用 gtk_main 函数以开始处理信号。第一步是初始化 GTK + 和创建应用窗口。

```
int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *table;

    /* --- GTK initialization --- */
    gtk_init (&argc, &argv);

    /* --- Create the calculator window --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

一般说来,你应该给应用窗口一个描述性的标题。还要给窗口定义最小的窗口尺寸。因为应用程序的缺省窗口尺寸太小,在这样小的窗口上,用户很难用鼠标单击窗口中的按钮。

```
    /* --- Give the window a title. --- */
    gtk_window_set_title (GTK_WINDOW (window), "Calculator");

    /* --- Set the window size. --- */
    gtk_widget_set_usize (window, 200, 200);
```

在通常情况下,应该监听关闭应用程序窗口的信号。同时,当用户决定使用键盘而不是鼠标的时候,应用程序窗口也应该能够响应按键(key-press)事件。它利用前面定义的表将击键映射到相应的按钮,就象用户在用鼠标点击按钮时一样处理键盘输入。

```
    /* --- We care if a key is pressed --- */
    gtk_signal_connect (GTK_OBJECT (window), "key_press_event",
                        GTK_SIGNAL_FUNC (key_press), NULL);

    /* --- You should always remember to connect the "delete_event"
     *      to the main window. --- */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (CloseAppWindow), NULL);
```

包含按钮的组合表是一个 5×5 的表,第一行用作显示屏幕。CreatCalculatorButtons 函数使用按钮数组,并将按钮放在表中正确的位置。同时此函数也要为所有按钮加一个通用的事件处理程序,对所有诸如鼠标点击按钮和按键操作作出响应。

```
/* --- Create a 5x5 table for the items in the calculator. --- */
table = gtk_table_new (5, 5, TRUE);

/* --- Create the calculator buttons. --- */
CreateCalculatorButtons (table);
```

显示屏仿照普通计算器的显示方式,它是控制正文右对齐的标号。显示屏幕放在组合表中的第一行,并横跨组合表中的所有列。

```
/* --- Create the calculator LED --- */
label = gtk_label_new ("0");
gtk_misc_set_alignment (GTK_MISC (label), 1, .5);

/* --- Add label to the table --- */
gtk_table_attach_defaults (GTK_TABLE (table), label,
                           0, 4, 0, 1);
```

下一步,使所有的构件成为可见,并加到主窗口中。

```
gtk_widget_show (label);

/* --- Make them visible --- */
gtk_container_add (GTK_CONTAINER (window), table);
gtk_widget_show (table);
gtk_widget_show (window);
```

最后,调用 gtk_main 函数以开始事件处理循环。

```
    gtk_main ();
    exit (0);
}
```

"detete_event"事件处理程序是必要的,它在关闭计算器时撤消 GTK+。

```
/*
 * CloseAppWindow
 *
 * The window is closing down, end the gtk loop
 */
gint CloseAppWindow (GtkWidget *widget, gpointer data)
{
    gtk_main_quit ();
```

```
    return (FALSE);
}
```

7.1.3 CreateCalculatorButtons

正象本章前面所讲过的，CreateCalculatorButtons 函数从按钮数组中建立按钮，使用在表中说明的标号、行和列创建每一个按钮。而 GtkWidget 代表最后被创建和保存在组合表中的按钮。实际上，CreateCalculatorButtons 函数依赖 CreateButton 函数来创建按钮。

```
void CreateCalculatorButtons (GtkWidget *table)
{
    int nIndex;

    /* --- Run through the list of buttons. --- */
    for (nIndex = 0; nIndex < nButtons; nIndex++) {

        /* --- Create a button and store the widget --- */
        buttonList[nIndex].widget =
                CreateButton (table,
                              buttonList[nIndex].szLabel,
                              buttonList[nIndex].row,
                              buttonList[nIndex].col);
    }
}
```

7.1.4 CreateButton

CreateButton 函数在组合表中创建一个带标号按钮。这个函数还要为 button _ clicked 事件设置事件处理程序。请注意，不同的按钮使用同一个 button _ clicked 回呼函数。同从发出点击事件信号的按钮中得到标号相比，更容易的方法是配置回呼函数将按钮的标号做为参数传递。当用 gtk _ signal _ connent 函数设置 button _ clicked 回呼函数时，按钮标号就作为 gpointer 数据值被传入。当在按钮上发生点击事件时，回呼函数将作为数据参数接受信息。

```
/*
 * CreateButton
 *
 * Create a button, assign event handlers, and attach the button to the
 * table in the proper place.
 */
GtkWidget *CreateButton (GtkWidget *table, char *szLabel, int row, int column)
{
    GtkWidget *button;
```

```
    /* --- Create the button --- */
    button = gtk_button_new_with_label (szLabel);

    /* --- We care if the button is clicked --- */
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (button_clicked), szLabel);

    /* --- Put the button in the table in the right place. --- */
    gtk_table_attach (GTK_TABLE (table), button,
                      column, column + 1,
                      row, row + 1,
                      GTK_FILL | GTK_EXPAND,
                      GTK_FILL | GTK_EXPAND,
                      5, 5);

    /* --- Make the button visible --- */
    gtk_widget_show (button);

    /* --- return the button. --- */
    return (button);
}
```

事件处理程序在处理鼠标和按键事件时必须能确定按下了哪一个按钮,并且根据此按钮更新计算器中的显示。对于点击鼠标事件,点击任何一个按钮,被点击的按钮就取得事件信号。而对于按键事件,得到事件的按钮并不一定是用户想要按下的按钮。例如,如果光标位于按钮"3"上,但用户在键盘上按下"5"键,按钮"3"就将得到按下"5"键的信号。一般情况下,对键的处理应该像 button_clicked 信号一样,按下哪一个按钮,该按钮就得到相应的信号(按下"5"按钮;"5"按钮就应该得到按下"5"键的信号)。此外,如果用户正在使用键盘,那么光标就应该转移到用户所敲击的那个按钮上。在前面的例子中,因为按下了"5"键,光标应该转移到按钮"5"上。这种操作给用户提供了额外的视觉反馈。这更象一个美学问题而不是一个编程问题。

不是将按键事件回呼函数放在按钮上,而是将它放在应用程序窗口中。这样,如果信号没有被光标所在的按钮处理,它就会落到应用程序窗口。应用程序窗口的按键事件回呼函数将根据被按下的键发送 button_clicked 信号。

点击鼠标事件和按键事件的处理是类似的。按键事件处理程序将从这个条件中得到好处。它可以调用同一个处理 button_clicked 信号的函数。

按键回呼函数是一个很小的函数,因为它只需要根据按下的键找出应该按下哪一个按钮。回呼函数在表中查找键,并试图将此键和按钮标号上的字符串匹配。这个过程只比较一位字符。它并不用在象 1/x 那样复杂的函数,因为这样的函数很难用一次按键来表达。它经常被用于计算器中的普通字符,例如数字和简单的操作符(+ - / *),当按钮被找到后,光标转移

到该按钮上,以提供视觉反馈。同时调用 gtk _ button _ clicked 函数,用来发出此按钮的点击(Clicked)信号 。

```
/* key_press
 *
 * Handle the button "key_press" event.
 *
 * Function looks for the keystroke in the calculator
 * data structure and (if a match is found) presses the
 * button that matches the keystroke for the user.  It
 * keeps our code small because we only have to handle the
 * button_clicked events.
 */
void key_press (GtkWidget *widget,
                GdkEventKey *event,
                gpointer data)
{
    int nIndex;

    /* --- Search through the buttons --- */
    for (nIndex = 0; nIndex < nButtons; nIndex++) {

        /* --- If the keystroke is the first character of a button AND --- */
        /* --- the button label length is one. --- */
        if (event->keyval == buttonList[nIndex].szLabel[0] &&
            buttonList[nIndex].szLabel[1] == (char) 0) {

            /* --- Set focus to that button --- */
            gtk_widget_grab_focus (buttonList[nIndex].widget);

            /* --- Make like the button was clicked to do processing. --- ? */
            gtk_button_clicked (GTK_BUTTON (buttonList[nIndex].widget));
            return;
        }
    }
}
```

如果按下一个按钮或从键盘上按下一个键,将调用 button _ clicked 事件处理程序。函数 button _ clicked 跟踪计算器的当前状态,执行运算或者只调用 gtk _ label _ set 函数在显示屏幕上显示数字。当然,在 button _ clicked 函数中可以加入更多种处理。例如,运算的优先级、括号、更多种计算(如 sin、cos、tan 等)、十六进制和二进制之间的转换等。这部分就留给读者作为练习。

函数 button_clicked 包括两种不同的情况。用户可以输入一个数字(0-9)或执行一种运算。如果用户按的是一个数字,通常表示需要将它加到显示中。例如,在显示屏上显示的是"45",这时用户又输入了一个"2",显示屏上应出现"452"。如果在输入"2"前还输入了一个二元操作符(+ - / *)则另当别论。当用户输入二元操作符时,显示器上仍应显示第一个数。当用户输入"2"时,将把它作为二元计算的第二个数的第一位数字输入到显示屏,完全替代第一个数。

第二种情况是用户输入一个操作符。如果输入了操作符,将同时使用在缓冲区中保存的数字将和显示器上的数字。例如,用户输入了"235 + 111",这时没有完成操作。但是,如果用户又输入了"=",计算器就去查询前一个命令"+"、缓冲区中的数字"235"和显示屏上的数字"111",并执行"235"和"111"的加运算。如果用户在输入"235 + 111"后又输入一个"+"而不是"=",则与前面所说的情况是一样的,唯一不同的是"+"号使计算器进入等待输入下一个数字的模式。

```
/*
 * button_clicked
 *
 * widget - button pressed.
 * data - button label.
 *
 * Button was pressed, handle it.
 */
void button_clicked (GtkWidget *widget, gpointer data)
{
    char ch = *((char *) data);
    int len;
    char *str;

    /* --- Get the button label --- */
    str = (char *) data;

    /* --- Entering a number... --- */
    if (FloatingPointChar (ch) && strlen (str) == 1) {

        HandleDigit (str, ch);

    } else {

        /* --- Clear? --- */
        if (strcmp (str, "CE") == 0) {
            gtk_label_set (GTK_LABEL (label), "0");
            return;
```

```
        /* --- BIG clear? --- */
        } else if (strcmp (str, "C") == 0) {
            prevCmd = (char) 0;
            lastChar = (char) 0;
            gtk_label_set (GTK_LABEL (label), "0");
            return;

        } else {

            /* --- Maybe it's a unary operator? --- */
            MaybeUnaryOperation (str);
        }

        /* --- See if there's a binary operation to do --- */
        HandleBinaryOperation ();

        prevCmd = ch;
    }
    lastChar = ch;
}
```

除非前面输入是运算命令，处理数字的程序将输入的数字加到显示的数字的后面。在输入命令的情况下，接着输入的数字将成为显示区中的第一个字符。

```
/*
 * HandleDigit
 *
 * Digit button was pressed, deal with it.  How it
 * is dealt with depends on the situation.
 */
void HandleDigit (char *str, char ch)
{
    char *labelText;
    char buffer[BUF_SIZE];
    int  len;

    /* --- And they just did a command --- */
    if (Command (lastChar)) {

        /* --- Clear the digit field --- */
        gtk_label_set (GTK_LABEL (label), "");
```

```
        /* --- If they did a computation --- */
        if (lastChar == '=') {

            /* --- Clear out the command --- */
            lastChar = (char) 0;
            prevCmd = (char) 0;
        }
    }

    /* --- Get the buffer in the LED --- */
    gtk_label_get (GTK_LABEL (label), &labelText);
    strcpy (buffer, labelText);

    /* --- Add the new character on it. --- */
    len = strlen (buffer);
    buffer[len] = (gchar) ch;
    buffer[len+1] = (gchar) 0;

    /* --- Trim leading zeros. --- */
    TrimLeadingZeros (buffer);

    /* --- Add digit to field. --- */
    gtk_label_set (GTK_LABEL (label), (char *) buffer);
}
```

计算器处理两种运算操作：一元运算和二元运算。MaybeUnaryOperation 函数判断运算是否是一元的。如果是，就执行一元运算。

```
/*
 * MaybeUnaryOperation
 *
 * str - label on button that describes operation.
 *
 * Check to see if the user hit a unary operator button -
 * like %, sqrt, 1/x, etc that should be dealt with NOW
 * not later.
 */
void MaybeUnaryOperation (char *str)
{
    char *labelText;
    char buffer[BUF_SIZE];
    float num2;
```

```
    /* --- Get number in the field. --- */
    gtk_label_get (GTK_LABEL (label), &labelText);
    num2 = atof (labelText);

    /* --- Percentage? --- */
    if (strcmp (str, "%") == 0) {
        num2 = num2 / 100;

    /* --- Trying for 1/x? --- */
    } else if (strcmp (str, "1/x") == 0) {

        /* --- Can't divide by zero. --- */
        if (num2 == 0) {
            return;
        }
        num2 = 1 / num2;

    /* --- Calculate sqrt --- */
    } else if (strcmp (str, "sqrt") == 0) {
        num2 = sqrt ((double) num2);

    /* --- Calculate square --- */
    } else if (strcmp (str, "x^2") == 0) {
        num2 = num2 * num2;
    }

    /* --- Put the number back. --- */
    sprintf (buffer, "%f", (float) num2);
    TrimTrailingZeros (buffer);
    TrimLeadingZeros (buffer);
    gtk_label_set (GTK_LABEL (label), buffer);
}
```

HandleBinaryoperation 函数检查是否需要完成二元运算。全局变量 num1 存储此计算中的第一个数，而第二个数则显示在 LED 屏幕上。当计算结束后，计算结果储存在全局变量 num1 中，因为它将是下一次二元计算的第一个数。

```
/*
 * HandleBinaryOperation
 *
 * Perform a calculation with two sets of digits.  One set
 * is in the LED display and the other set is saved in
```

```
 * a global variable (num1).
 */
void HandleBinaryOperation ()
{
    char buffer[BUF_SIZE];
    char *labelText;
    float num2;

    /* --- Get number in the field. --- */
    gtk_label_get (GTK_LABEL (label), &labelText);
    num2 = atof (labelText);

    /* --- Calculate based on previous command. --- */
    switch (prevCmd) {
        case '+':
            num1 = num1 + num2;
            break;

        case '-':
            num1 = num1 - num2;
            break;

        case '*':
            num1 = num1 * num2;
            break;

        case '/':
            num1 = num1 / num2;
            break;

        case '=':
            num1 = num2;
            break;

        default:
            num1 = num2;
            break;
    }

    /* --- Put the number back. --- */
    sprintf (buffer, "%f", (float) num1);
    TrimTrailingZeros (buffer);
```

```
    TrimLeadingZeros (buffer);
    gtk_label_set (GTK_LABEL (label), buffer);
}
```

其余的函数是一些辅助函数。合理选择它们的名称有助于提高程序的可读性。例如，FloatingPointChar 函数判断字符是否是浮点字符(0-9 或者“.”)。如果一行代码写成：

```
if (FloatingPointChar (ch)) {
```

它显然要比写成：

```
if (isdigit (ch) || ch == ".") {
```

容易阅读。

```
/*
 * TrimTrailingZeros
 *
 * Get rid of trailing zeros
 * Takes the string and removes the trailing zeros.
 */
void TrirTrailingZeros (char *szDigits)
{
    int nIndex;
    int bDecimal = FALSE;
    int nPos = -1;

    /* --- Loop through the string. --- */
    for (nIndex = 0; nIndex < strlen (szDigits); nIndex++) {

        /* --- Is this a decimal? --- */
        if (szDigits[nIndex] == '.') {
            bDecimal = TRUE;
        }

        /* --- If we're on the right side of the decimal... --- */
        if (bDecimal) {

            /* --- A zero?  Hmm... from this point on? --- */
            if (szDigits[nIndex] == '0') {

                /* --- If we don't have a point yet... --- */
                if (nPos < 0) {
```

```
                /* --- Save this as a point. --- */
                nPos = nIndex;
            }
        } else {

            /* --- Clear it.  Bad point. --- */
            nPos = -1;
        }
    }

    /* --- Truncate the field. --- */
    if (nPos > 0) {
        szDigits[nPos] = (char) 0;
    }
}
/*
 * TrimLeadingZeros
 *
 * Trim the leading zeros.
 *
 * Converts numbers like "0000012" to "12"
 */
void TrimLeadingZeros (char * szDigits)
{
    int nPos;

    if (szDigits == NULL) return;

    /* --- While we have a combination, a digit in front --- */
    for (nPos = 0; (szDigits[nPos] && szDigits[nPos] == '0'); nPos++) {

        /* --- If the digit is a zero and next char is a digit --- */
        if (isdigit (szDigits[nPos + 1])) {

            /* --- Blank the field. --- */
            szDigits[nPos] = ' ';
        }
    }
}
/*
 * Command
```

```
 *
 * Returns true if the character is a two-digit command.
 */
int Command (char ch)
{
    switch (ch) {
        case '+':
        case '-':
        case '/':
        case '*':
        case '=':
            return (TRUE);
    }
    return (FALSE);
}



/*
 * FloatingPointChar
 *
 * Returns true if the character is any of [0123456789.]
 */
int FloatingPointChar (char ch)
{

    return (isdigit (ch) || ch == '.');
}
```

7.2　小结

利用按钮和标号构件建立简单的计算器是很容易的。将简单的事件处理程序与构件结合起来，你就得到了一个应用程序。你已经为编写大的应用程序作好了必要的准备。

第 8 章　开发简单的字处理程序

本章将重点放在利用前面讨论的内容开发简单的字处理程序，它类似于 Windows 系统中的 Notepad。用已有的应用作样板进行仿制远比从头设计要容易得多。Windows 的 Notepad 允许用户编辑和存储简单的文本文件，它的能力有限。但是，从另一个角度看，将几个构件结合在一起建立小而稳定的应用，它仍然不失为一个好的例子。

从构件组成完整的应用程序，Notepad 应用程序(图 8.1)说明许多基本概念：包括菜单、工具条、各种对话框以及用于编辑信息的区域。此外，Notepad 与 Microsoft Windows 提供的其他应用结合得不太紧密，你可以根据自己的需要定制它。也可以只将它当作简捷的编辑程序使用。

它的主菜单包括“File”、“Edit”、“Search”和“Help”等菜单项。在“File”菜单下有“New”、“Open”、“Save”、“Save As…为”以及“Quit”等选项。“New”选项建立新文档，“Open”选项装入已有文档，“Save”选项保存当前文档，“Save As…为”选项将当前文档用其他名称保存，而“Quit”选项则退出应用程序。“Edit”菜单包括“Cut”、“Copy”以及“Paste”等剪贴板命令。“Search”菜单下只有“Find”选项，而“Help”菜单下只有用来显示作者及版本信息的“About…”选项。

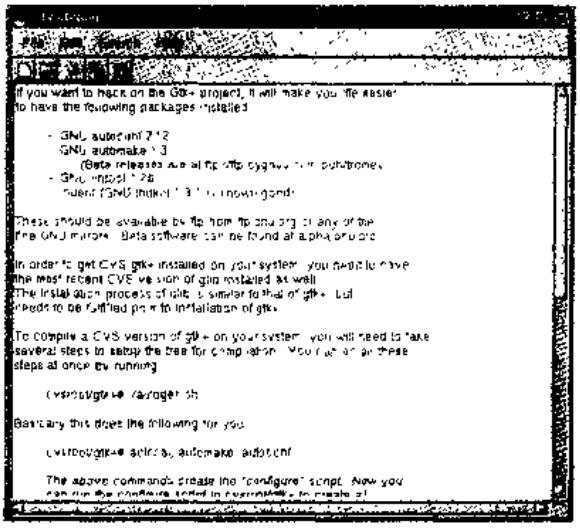


图 8.1　Gnotepad

除了菜单以外，应用程序的工具条使用户更快捷地使用大部分命令。Notepad 本来不带工具条。在这儿加上它增加了应用的功能，而且也不是件难事。

8.1 main.c

本程序和大多数 GTK + 应用程序一样开始:创建窗口,给一个标题,并创建纵向组合框。将菜单和工具条构件放在组合框的上面,下面接着放显示和编辑文件的正文构件。将用于创建菜单和工具条的程序封装在 CreateMenu 函数中,而将建立实际的正文构件的程序放在 CreateText 函数中。

在建立应用程序的工具条时存在一个问题。建立工具条图标需要将像元图正文数据转换为 GdkPixmap。这样,它才能被转换为像元图构件。这种转换需要 X Window 作为其中一个参数,你可以在使用 gtk _ realize _ widget 函数进行配置时使窗口成为不可见,而不必在此时使窗口成为可见(这样作需要建立 X Window),然后加载和显示工具条。gtk _ realize _ widget 函数为构件建立 X Window 面不使窗口可见。通常,这部份放在 gtk _ widget _ show 函数中实现,但在这儿你需要在使应用的窗口可见之前建立 X Window。你可以不用 gtk _ realize _ widget 函数而将 CreateMenu 函数放在窗口可见之后,但这可不是一个好方法,因为窗口会在还没有完成前就被显示(比方说,没有菜单和工具条)。

```
/*
 * --- main
 *
 * program begins here.
 */
int main(int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *main_vbox;

    /* --- Start up GTK --- */
    gtk_init(&argc, &argv);

    /* --- Create the top-level window --- */
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

    /* --- Title and border --- */
    gtk_window_set_title (GTK_WINDOW (window), "gnotepad");
    gtk_container_border_width (GTK_CONTAINER (window), 0);

    /* --- Listen for the main window being closed --- */
    gtk_signal_connect (GTK_OBJECT(window), "delete_event",
              GTK_SIGNAL_FUNC (ClosingAppWindow),
              NULL);
```

```
    /* --- Set the window title and size --- */
    gtk_widget_set_usize (GTK_WIDGET(window), 200, 200);

    /* --- Create a new vertical box for storing widgets --- */
    main_vbox = gtk_vbox_new (FALSE, 1);

    /* --- border width is 1 --- */
    gtk_container_border_width (GTK_CONTAINER(main_vbox), 1);

    /* --- Add the vertical box to the main window --- */
    gtk_container_add (GTK_CONTAINER(window). main_vbox);

    /* --- Make the vertical box visible --- */
    gtk_widget_show (main_vbox);

    /* --- Associate window with X Window to create pixmaps. --- */
    gtk_widget_realize (window);

    /* --- Create the menu/toolbar. --- */
    CreateMenu (window, main_vbox);
    CreateText (window, main_vbox);

    /* --- Show the top-level window --- */
    gtk_widget_show (window);

    /* --- Loop and process messages --- */
    gtk_main();

    return(0);
}
```

8.2　menu.c

本文件包括定义工具条图标和建立菜单和工具条按钮的程序。大部份代码和以前建立的简单的界面是相似的。

在 menu.c 中的第一段代码为工具条定义了它的位图。位图应该清楚而容易辨认。总的来说，你应该尽量减少位图中颜色的数量，少到 4 种或更少。位图是为了使程序的功能容易进行识别，太多的颜色就违反了建立位图的初衷。其实大部份应用程序使用的图标都很相似，你也应尽量和它们一致，因为最容易使用户感到遗憾便是使用一组新的图标。这里我们就沿用了前面章节中使用过的图标。

从主程序中调用的 CreateMenu 代码用来设置快捷键表和菜单,并完成对 CreateToolbar 函数的调用以建立工具条。每个菜单项的回呼函数名称前都用菜单项作为前缀,可以使这些函数与普通函数相区别。例如,当 File->New 被选中时,menu_New 函数就会被调用。

```
/*
 * CreateMenu
 *
 * Create the main window and the menu/toolbar associated with it
 */
void CreateMenu (GtkWidget *window, GtkWidget *vbox_main)
{
    GtkWidget *menubar;
    GtkWidget *menu;
    GtkWidget *menuitem;

    win_main = window;

    /* --- Create accel table --- */
    accel_group = gtk_accel_group_new ();
    gtk_accel_group_attach (accel_group, GTK_OBJECT (window));

    /* --- Menu bar --- */
    menubar = gtk_menu_bar_new ();
    gtk_box_pack_start (GTK_BOX (vbox_main), menubar, FALSE, TRUE, 0);
    gtk_widget_show (menubar);

    /* ------------------
       --- File menu ---
       ------------------ */
    menu = CreateBarSubMenu (menubar, "File");

    menuitem = CreateMenuItem (menu, "New","^N",
                    "Create a new item",
                    GTK_SIGNAL_FUNC (menu_New), "new");

    menuitem = CreateMenuItem (menu, "Open","^O",
                    "Open an existing item",
                    GTK_SIGNAL_FUNC (menu_Open), "open");

    menuitem = CreateMenuItem (menu, "Import RTF","",
                    "Import RTF file",
                    GTK_SIGNAL_FUNC (menu_ImportRTF), "import rtf");
```

```
menuitem = CreateMenuItem (menu, "Save","^S",
                "Save current item",
                GTK_SIGNAL_FUNC (menu_Save), "Save");

menuitem = CreateMenuItem (menu, "Save As...", " ",
                "Save current item with new name",
                GTK_SIGNAL_FUNC (menu_SaveAs), "Save as");

menuitem = CreateMenuItem (menu, NULL, NULL,
                NULL, NULL, NULL);

menuitem = CreateMenuItem (menu, "Quit","" ,
                "What's more descriptive than quit?",
                GTK_SIGNAL_FUNC (menu_Quit), "quit");

/* ----------------
   --- Edit menu ---
   ---------------- */
menu = CreateBarSubMenu (menubar, "Edit");

menuitem = CreateMenuItem (menu, "Cut","^X",
                "Remove item and place into clipboard",
                GTK_SIGNAL_FUNC (menu_Cut), "cut");

menuitem = CreateMenuItem (menu, "Copy","^C",
                "Place a copy of the item in the clipboard",
                GTK_SIGNAL_FUNC (menu_Copy), "copy");

menuitem = CreateMenuItem (menu, "Paste","^V",
                "Paste item",
                GTK_SIGNAL_FUNC (menu_Paste), "paste");


/* --------------------
   --- Find submenu ---
   -------------------- */
menu = CreateBarSubMenu (menubar, "Search");

menuitem = CreateMenuItem (menu, "Find","^F",
                "Find item",
                GTK_SIGNAL_FUNC (menu_Find), "find");
```

```
    /* ---------------------
       --- Help menu ---
       --------------------- */
    menu = CreateBarSubMenu (menubar, "Help");

    menuitem = CreateMenuItem (menu, "About", "",
                    "About",
                    GTK_SIGNAL_FUNC (menu_About), "about");

    /* --- Create the toolbar --- */
    CreateToolbar (vbox_main);
}
```

CreatMenu 函数的最后一部份是用于生成工具条的 CreateToolbar 函数。工具条上的大部份图标都沿用了前面讲的建立图形界面的例子中的图标,按钮对应的函数和菜单中的相应选项对应的函数是一样的。事实上,当按钮 New 被按下,GTK + 会去调用 menu_New 函数,这和选择了菜单项中的 File- > new 所调用的函数完全一样。

```
/*
 * CreateToolbar
 *
 * Create the toolbar for commonly used options
 */
void CreateToolbar (GtkWidget *vbox_main)
{

    /* --- Create the toolbar and add it to the window --- */
    toolbar = gtk_toolbar_new (GTK_ORIENTATION_HORIZONTAL, GTK_TOOLBAR_ICONS);
    gtk_box_pack_start (GTK_BOX (vbox_main), toolbar, FALSE, TRUE, 0);
    gtk_widget_show (toolbar);

    /*              ----
     * --- Create | new | button
     *              ----
     */
    gtk_toolbar_append_item (GTK_TOOLBAR (toolbar),
                      NULL, "New window", NULL,
                    CreateWidgetFromXpm (win_main, (gchar **) xpm_new),
                      (GtkSignalFunc) menu_New,
                           NULL);
```

```
/*              -----
 * --- Create ¦ open ¦ button
 *              -----
 */
gtk_toolbar_append_item (GTK_TOOLBAR (toolbar),
                "Open Dialog","Open dialog", " ",
                CreateWidgetFromXpm (win_main, (gchar ** ) xpm_open),
                  (GtkSignalFunc) menu_Open,
                        NULL);

/* --- Little gap --- */
gtk_toolbar_append_space (GTK_TOOLBAR (toolbar));

/*              -----
 * --- Create ¦ cut ¦ button
 *              -----
 */
gtk_toolbar_append_item (GTK_TOOLBAR (toolbar),
                  "Cut","Cut","",
                CreateWidgetFromXpm (win_main, (gchar ** ) xpm_cut),
                  (GtkSignalFunc) menu_Cut,
                        NULL);

/*              -----
 * --- Create ¦ copy ¦ button
 *              -----
 */
gtk_toolbar_append_item (GTK_TOOLBAR (toolbar),
                "Copy","Copy", " ",
                CreateWidgetFromXpm (win_main, (gchar ** ) xpm_copy),
                  (GtkSignalFunc) menu_Copy,
                        NULL);

/*              -----
 * --- Create ¦ paste ¦ button
 *              -----
 */
gtk_toolbar_append_item (GTK_TOOLBAR (toolbar),
                  "Paste","Paste"," " ,
                CreateWidgetFromXpm (win_main, (gchar ** ) xpm_paste),
                  (GtkSignalFunc) menu_Paste,
                        NULL);
}
```

8.3 misc.c

这个文件包含了由 CreateMenu 函数和 CreatToolbar 函数调用的一些低级函数。文件中没有增加新的内容，而只是收集了前面例子中有关菜单和工具条的部份。用于创建位图、菜单和子菜单的代码可以很容易地在这里重新使用。

8.4 showmessage.c

应用程序中有些地方需要向用户显示信息。为防止代码的重复，单独设了名为 showmessage 的函数来创建对话框。此函数还要调用名为 gtk_grab_add 函数以确保用户确实看到了弹出的对话框。

gtk_grab_add 函数将一直阻止用户完成其他功能，直至用户取消对话框。在用户按下 OK 按钮后，在 destroy 回呼函数中使用 gtk_grab_remove 函数释放对话框的独占状态。因为对话框可能在没有按 OK 或 Cancel 按钮的情况下被释放，所以将这段代码放在 destroy 回呼函数中。

函数 ShowMessage 显示一个标题和一段信息。它将标题放在对话框窗口上，而将信息放在带 OK 按钮的对话框中。

```
/*
 * ShowMessage
 *
 * Show a popup message to the user.
 */
void ShowMessage (char *szTitle, char *szMessage)
{
    GtkWidget *label;
    GtkWidget *button;
    GtkWidget *dialog_window;

    /* --- Create a dialog window --- */
    dialog_window = gtk_dialog_new ();

    gtk_signal_connect (GTK_OBJECT (dialog_window), "destroy",
            GTK_SIGNAL_FUNC (ClearShowMessage),
            NULL);

    /* --- Set the title and add a border --- */
    gtk_window_set_title (GTK_WINDOW (dialog_window), szTitle);
    gtk_container_border_width (GTK_CONTAINER (dialog_window), 0);
```

```
    /* --- Create an "OK" button with the focus --- */
    button = gtk_button_new_with_label ("OK");

    gtk_signal_connect (GTK_OBJECT (button), "clicked",
              GTK_SIGNAL_FUNC (CloseShowMessage),
              dialog_window);

    /* --- Default the "OK" button --- */
    GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);
    gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog_window)->action_area),
               button, TRUE, TRUE, 0);
    gtk_widget_grab_default (button);
    gtk_widget_show (button);

    /* --- Create a descriptive label --- */
    label = gtk_label_new (szMessage);

    /* --- Put some room around the label text --- */
    gtk_misc_set_padding (GTK_MISC (label), 10, 10);

    /* --- Add label to designated area on dialog box --- */
    gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog_window)->vbox),
              label, TRUE, TRUE, 0);

    /* --- Show the label --- */
    gtk_widget_show (label);


    /* --- Show the dialog box --- */
    gtk_widget_show (dialog_window);

    /* --- Only this window can have actions done. --- */
    gtk_grab_add (dialog_window);
}
```

为了使 ShowMessage 函数工作得更完美，有两种情况需要通过事件加以处理。用户可能点击 OK 按钮，也可能只是关闭了对话框。这些操作将产生不同的事件。点击按钮的操作会产生点击事件，但仅此而已，不再做其它事情。而关闭窗口将产生 destroy 事件，它不消除调用 gtk_grab_add 函数产生的结果。为了在用户点击 OK 按钮时关闭窗口，对 OK 按钮的点击事件设置为：一旦产生点击事件，就去调用 CloseShowMessage 函数。在此函数中调用 gtk_grab_remove，并删除对话框窗口。注意，对话框构件作为函数的数据参数被传送。这是因为在用 gtk_signal_connect 函数设置单击事件时，对话框窗口被作为数据参数传送。

```
/*
 * --- CloseShowMessage
 *
 * Routine to close the about dialog window.
 */
void CloseShowMessage (GtkWidget *widget, gpointer data)
{
    GtkWidget *dialog_widget = (GtkWidget *) data;

    /* --- Close the widget --- */
    gtk_widget_destroy (dialog_widget);
}
```

如果用户不是通过点击 OK 按钮来关闭窗口,那么释放对话框是在发出 destroy 信号之后的事。ClearShow Message 函数清除掉所有在创建对话框时生成的设置。但是在这里,它只调用了 gtk _ grab _ remove 函数。

```
/*
 * ClearShowMessage
 *
 * Release the window "grab"
 * Clear out the global dialog_window since that
 * is checked when the dialog box is brought up.
 */
void ClearShowMessage (GtkWidget *widget, gpointer data)
{
    gtk_grab_remove (widget);
}
```

8.5 about.c

你可以在应用程序中用 Show Message 程序段来实现简单的对话框。例如,菜单中用于显示应用程序有关信息的"About…"选项就是显示在对话框中。有关的信息通常包括程序所属的公司,它的作者以及版本号。整段"About…"代码所作的只是调用 Show Message 函数来显示信息。

```
/*
 * ShowAbout
 *
 * Show the about dialog box.  Reuse existing code.
 */
```

```
void ShowAbout ()
{
    ShowMessage ("About...",
                 "GtkNotepad v.07 \n -"
                 "Eric Harlow \n");

}
```

8.6 filesel.c

文件选择对话框是由打开和保存文件子程序所调用。为了提示用户其正在完成的任务，将对话框的标题发送给 GetFilename 函数。此外，为了使对话框具有一般性，还将一个函数送入 GetFilename。如果用户点击 OK 按钮，用被选择的文件名调用这个函数。用这种方法可以使同一段文件选择程序同时用于装入文件和保存文件。

此外，gtk_grab_add 函数将一直阻止用户完成其他功能，直至用户取消对话框。通过捕获 destroy 信号，集中完成清除工作。

文件选择的代码依赖于 GetFilename 分配的结构，这个结构被送到对话框的所有回呼函数中去。这段代码包含有关对话框构件的信息，以及用户点击 OK 按钮后程序要执行的操作。结构定义如下。

```
typedef struct {

    void ( * func) (gchar * );
    GtkWidget * filesel;

} typFileSelectionData;

/*
 * GetFilename
 *
 * Show a dialog box with a title and if "OK" is selected
 * call the function with the name of the file.
 */
void GetFilename (char * sTitle, void ( * callback) (char * ))
{
    GtkWidget * filew = NULL;
    typFileSelectionData * data;

    /* --- Create a new file selection widget --- */
    filew = gtk_file_selection_new (sTitle);
```

```
    data = g_malloc (sizeof (typFileSelectionData));
    data->func = callback;
    data->filesel = filew;

    gtk_signal_connect (GTK_OBJECT (filew), "destroy",
            (GtkSignalFunc) destroy, data);

    /* --- Connect the "OK" button --- */
    gtk_signal_connect (GTK_OBJECT
            (GTK_FILE_SELECTION (filew)->ok_button),
            "clicked", (GtkSignalFunc) FileOk, data);

    /* --- Connect the Cancel button --- */
    gtk_signal_connect_object (
            GTK_OBJECT (GTK_FILE_SELECTION (filew)->cancel_button),
            "clicked", (GtkSignalFunc) gtk_widget_destroy,
            (gpointer) filew);

    if (sFilename) {

        /* -  Set the default filename --- */
        gtk_file_selection_set_filename (GTK_FILE_SELECTION (filew),
                                         sFilename);
    }

    /* --- Show the dialog box --- */
    gtk_widget_show (filew);

    /* --- Grab the focus. --- */
    gtk_grab_add (filew);
}
```

如果用户按下 OK 按钮,传入 GetFilename 中的回呼函数就要被执行,它要用到从选择对话框中得到的文件名。对于该应用程序,传入的函数既有可能是为了装入文件也有可能是保存文件。这段代码中用了一个全局变量(sFilename)来保存选择的文件名。如果用户想用"Save As"选项来保存文件,那么原先选择的文件名将被显示在屏幕上,用户可以改变它。

用户点击 OK 按钮后,FileOk 函数被调用。它从构件中得到文件名,再将文件名传入另一个函数来完成工作,最后删除构件。其中被调用的函数取决于用户调用 GetFilename 时使用的函数,这个函数被保存在一个全局变量中。

这种方法有可能实现是因为在任何时候都只能有一个文件选择对话框被显示——这是由 gtk_grab_add 函数所保证的条件。你不能在没有关闭现有文件选择对话框的时候就打开另

一个文件选择对话框。

```
/*
 * FileOk
 *
 * The "OK" button has been clicked.
 * Call the function (func) to do what is needed
 * to the file.
 */
void FileOk (GtkWidget *w, gpointer data)
{
    char *sTempFile;
    typFileSelectionData *localdata;
    GtkWidget *filew;

    localdata = (typFileSelectionData *) data;
    filew = localdata->filesel;

    /* --- Which file? --- */
    sTempFile = gtk_file_selection_get_filename (
                            GTK_FILE_SELECTION (filew));

    /* --- Free old memory --- */
    if (sFilename) g_free (sFilename);

    /* --- Duplicate the string --- */
    sFilename = g_strdup (sTempFile);

    /* --- Call the function that does the work. --- */
    (*(localdata->func)) (sFilename);

    /* --- Close the dialog box--- */
    gtk_widget_destroy (filew);
}
```

destroy 函数通过调用 gtk_grab_remove 函数完成删除工作。对于对话框来说，这个函数是很常见的。

```
/*
 * destroy
 *
 * Function to handle the destroying of the dialog box. We must
 * release the focus that we grabbed.
```

```
 */
static void destroy (GtkWidget *widget, gpointer *data)
{
    /* --- Remove the focus. --- */
    gtk_grab_remove (widget);

    g_free (data);
}
```

8.7 Notepad.c

Notepad.c 中的函数主要与正文构件(TextWidget)打交道。在这里,有将文件打开并装入正文构件中的函数,有将正文构件中的内容保存到文件的函数,还有处理的剪切和粘贴的函数。

从 main.c 调用 CreateText 函数,它创建和配置一个在编辑器中使用的构件。正文构件和横向、纵向滚动条一起被放在 2 乘 2 的组合表中。将正文构件保存在静态变量中,允许这一模块中的函数访问它。模块以外的函数不能对此构件进行直接访问。

```
/*
 * CreateText
 *
 * Creates the text widget for the editor.
 *
 */
void CreateText (GtkWidget *window, GtkWidget *container)
{
    GtkWidget *table;
    GtkWidget *hscrollbar;
    GtkWidget *vscrollbar;

    /* --- Create a table to put text widget and scrollbars --- */
    table = gtk_table_new (2, 2, FALSE);

    /* --- Add table to container --- */
    gtk_container_add (GTK_CONTAINER (container), table);

    /* --- No gaps so the scrollbars look like they are a
           part of the widget. --- */
    gtk_table_set_row_spacing (GTK_TABLE (table), 0, 2);
    gtk_table_set_col_spacing (GTK_TABLE (table), 0, 2);
```

```
/*  -- Add packing table to the container and make visible --- */
gtk_widget_show (table);

/* --- Create the text widget --- */
text = gtk_text_new (NULL, NULL);

/* --- Allow it to be edited --- */
gtk_text_set_editable (GTK_TEXT (text), TRUE);

/* --- Insert the text widget into the table --- */
gtk_table_attach (GTK_TABLE (table), text, 0, 1, 0, 1,
        GTK_EXPAND | GTK_SHRINK | GTK_FILL,
        GTK_EXPAND | GTK_SHRINK | GTK_FILL, 0, 0);

/* --- Make it visible --- */
gtk_widget_show (text);

/* --- Add a horizontal scrollbar --- */
hscrollbar = gtk_hscrollbar_new (GTK_TEXT (text)->hadj);
gtk_table_attach (GTK_TABLE (table), hscrollbar, 0, 1, 1, 2,
        GTK_EXPAND | GTK_FILL | GTK_SHRINK, GTK_FILL, 0, 0);
gtk_widget_show (hscrollbar);

/* --- Add a vertical scrollbar --- */
vscrollbar = gtk_vscrollbar_new (GTK_TEXT (text)->vadj);
gtk_table_attach (GTK_TABLE (table), vscrollbar, 1, 2, 0, 1,
        GTK_FILL, GTK_EXPAND | GTK_SHRINK | GTK_FILL, 0, 0);
gtk_widget_show (vscrollbar);
```

8.7.1　Cut、Copy 和 Paste

正文构件本身就支持剪贴版的“Cut”、“Copy”和“Paste”命令。但通常用户不能使用键盘(Ctrl + X, Ctrl + C, Ctrl + V)来完成这些操作。从用户的角度看,菜单和工具条中的剪贴板操作应该同快捷键完整地结合在一起。菜单和工具条中的命令映射到一系列的函数,它们和键盘命令完成同样的工作。例如,在菜单中的“Cut”选项可以通过调用 gtk_editable_cut_clipboard 函数删除文本并把它放入剪切版。

```
/*
 * TextCut
 *
```

```
 * Cut the selected text from the text widget and
 * put it in the clipboard.
 */
void TextCut (GtkWidget *widget, gpointer data)
{
    gtk_editable_cut_clipboard (GTK_EDITABLE (text));
}
```

事实上,我们可以一样容易地实现复制和粘贴命令。

```
/*
 * TextCopy
 *
 * Copy the selected text in the widget into the
 * clipboard.
 */
void TextCopy (GtkWidget *widget, gpointer data)
{
    gtk_editable_copy_clipboard (GTK_EDITABLE (text));
}


/*
 * TextPaste
 *
 * Paste the text in the clipboard into the text widget
 */
void TextPaste (GtkWidget *widget, gpointer data)
{
    gtk_editable_paste_clipboard (GTK_EDITABLE (text));
}
```

8.7.2 建立空白构件

现在编辑器已经具有剪切、复制和粘贴的功能。正文构件还要处理一些其他的命令。菜单中"File->New"选项应该将构件中的全部正文清空,这项操作可以用一步完成。

```
/*
 * ClearFile
 *
 * Called to clear all the text in the text widget
 */
void ClearFile (GtkWidget *widget, gpointer data)
{
```

```
    gtk_editable_delete_text (GTK_EDITABLE (text), 0, -1);
}
```

8.7.3　装入文件

打开(或装入)文件需要用到文件选择对话框。将 LoadFile 函数传送给 GetFielname 函数。如果用户点击 OK 按钮,将调用此函数。

```
/*
 * menu_Open (main.c)
 *
 * Give the user a dialog box to read a file.
 */
void menu_Open (GtkWidget *widget, gpointer data)
{
    GetFilename ("Open", LoadFile);
}
```

LoadFile 函数在用进度条显示进度的同时将文件读入构件。有三个进度条的函数被用来显示装入文件的进度。StartProgress 用于创建进度条,UpdateProgress 用于更新进度,而 EndProgress 将进度条关闭。使用 gtk_text_freeze 函数防止构件在装载文件的过程中花时间去刷新正文。读入文件需要用到一个长度为 BUF_SIZE 的缓冲区。常量 BUF_SIZE 可以调整,它的值越大,装载就越快。在这里,我们设了一个小一点儿的值,不然文件装载的速度就会太快以致于都来不及显示进度条。

```
/*
 * LoadFile
 *
 * sFilename - file to load
 *
 * Reads in the file and places it in the text widget.
 */
void LoadFile (char *sFilename)
{
    char buffer[BUF_SIZE];
    int nchars;
    FILE *infile;
    struct stat fileStatus;
    long fileLen = 0;

    /* --- Freeze the widget --- */
    gtk_text_freeze (GTK_TEXT (text));
```

```
    /* --- Empty the widget --- */
    gtk_editable_delete_text (GTK_EDITABLE (text), 0, -1);

    /* --- Get file information --- */
    stat (sFilename, &fileStatus);
    fileLen = fileStatus.st_size;

    StartProgress ();

    /* --- Open the file --- */
    infile = fopen (sFilename, "r");

    /* --- If we open the file. --- */
    if (infile) {

        /* --- Read a chunk... --- */
        while ((nchars = fread (buffer, 1, BUF_SIZE, infile)) > 0) {

            /* --- Update the progress bar --- */
            UpdateProgress (ftell (infile), fileLen);

            /* --- Insert the text --- */
            gtk_text_insert (GTK_TEXT (text), NULL, NULL,
                  NULL, buffer, nchars);

            /* --- Less than buffer means EOF --- */
            if (nchars < BUF_SIZE)
                break;
        }

        /* --- Close file --- */
        fclose (infile);
    }
    EndProgress ();

    /* --- Unfreeze the text widget - it'll repaint now --- */
    gtk_text_thaw (GTK_TEXT (text));
}
```

8.7.4 保存文件

保存文件和打开文件差不多,不同的是文件可以使用一大块内存写出。用 gtk_editable_

get _ chars 函数在构件中取得的内存应该用 g _ free 函数加以释放。

```
/*
 * SaveFile
 *
 * sFilename - filename to load
 *
 * Save the file.
 */
void SaveFile (char * sFilename)
{
    FILE * outfile;
    char * buffer;
    int nchars;

    gtk_ text _ freeze (GTK_ TEXT (text));

    /* --- Open the file --- */
    outfile = fopen (sFilename, "w");

    if (outfile) {

        /* --- Get the string from the widget --- */
        buffer = gtk_ editable _ get _ chars (
                    GTK_ EDITABLE (text),
                    (gint) 0,
                    (gint) gtk_ text _ get _ length (GTK_ TEXT (text)));

        /* --- Write the buffer to disk --- */
        nchars = fwrite (buffer, sizeof (char), strlen (buffer), outfile);

        /* --- Close the file --- */
        fclose (outfile);

        if (nchars ! = strlen (buffer)) {

            ShowMessage ("Save","Error: Can't write file.");
        }

        /* --- Free up the memory --- */
        g _ free (buffer);
```

```
    } else {

        ShowMessage ("Save","Error: Can't save file.");
    }

    gtk_text_thaw (GTK_TEXT (text));
}
```

8.7.5 搜索正文

搜索功能的实现用到了非独占对话框。不同于"About…"对话框和选择文件对话框,当搜索对话框开着时,允许用户在应用中进行其他操作。它也允许其他窗口在前端,但不论什么时候,搜索对话窗口都只能有一个。

"Find"菜单选项映射到 menu_Find 函数,此函数创建对话框,并允许用户从当前光标位置到文件结束的范围内查询字符串。传入对话框的两个函数分别对应于对话框中的两个按钮。如果用户点击"Find Next"按钮,将调用 FindFunction 函数在构件中查找字符串。如果点击"Cancel"按钮,将调用关闭窗口的 CancelFunction 函数。

```
/*
 * menu_Find
 *
 * Find a string in the editor
 */
void menu_Find (GtkWidget *widget, gpointer data)
{
    FindStringDialog ("Find", FindFunction, CancelFunction);
}
```

FindStringDialog 函数创建对话框,并将 FindFunction 和 CancelFunction 分别分配给"Find Next"和"Cancel"按钮。在创建对话框以前,此函数必先对全局变量(dialog_window)进行检查,以确保只有一个对话框是打开的。

```
/*
 * FindStringDialog
 *
 * Function to display a search dialog box
 */
void FindStringDialog (char *szMessage, void (*YesFunc)(), void (*NoFunc)())
{
    GtkWidget *label;
    GtkWidget *button;
    GtkWidget *hbox;
```

```
/* --- If window is already open, just return --- */
if (dialog_window) return;

/* --- Create the dialog box--- */
dialog_window = gtk_dialog_new();

gtk_signal_connect (GTK_OBJECT (dialog_window), "destroy",
                    GTK_SIGNAL_FUNC (CloseFindDialog),
                    dialog_window);

/* --- Set the title --- */
gtk_window_set_title (GTK_WINDOW (dialog_window), "Find");

/* --- Add a small border --- */
gtk_container_border_width (GTK_CONTAINER (dialog_window), 5);

/*
 * --- Create the message
 */

hbox = gtk_hbox_new (TRUE, TRUE);

/* --- Create a label with the message --- */
label = gtk_label_new ("Find What:");
gtk_widget_show (label);

/* --- Create the entry field --- */
entry = gtk_entry_new();
gtk_widget_show (entry);

/* --- If they're searched before --- */
if (szNeedle) {

    /* --- Prefill with the value they looked for last time --- */
    gtk_entry_set_text (GTK_ENTRY (entry), szNeedle);
}

gtk_box_pack_start (GTK_BOX (hbox),
                    label, TRUE, TRUE, 0);

gtk_box_pack_start (GTK_BOX (hbox),
                    entry, TRUE, TRUE, 0);
```

```
gtk_widget_show (hbox);

/* --- Add label to the dialog box--- */
gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog_window)->vbox),
                    hbox, TRUE, TRUE, 0);

/* --- Create the "yes" buttor --- */
button = gtk_button_new_with_label ("Find Next");

gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (YesFunc),
                    dialog_window);

/*      Add the button to the dialog box--- */
gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog_window)->action_area),
                    button, TRUE, TRUE, 0);

/* --- Make the button visible -- · */
gtk_widget_show (button);

/*
 * --- Cancel button
 */

/* --- Create the "Cancel" button --- */
button = gtk_button_new_with_label ("Cancel");

gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (NoFunc),
                    dialog_window);

/* --- Allow "Cancel" to be a default --- */
GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);

/* --- Add the "Cancel" buttor to the dialog box--- */
gtk_box_pack_start (GTK_BOX
              (GTK_DIALOG (dialog_window)->action_area),
                 button, TRUE, TRUE, 0);

/* --- Make the "Cancel" the default --- */
gtk_widget_grab_default (button);
```

```
    /* --- Make the button visible. --- */
    gtk_widget_show (button);

    /* --- Show the dialog box--- */
    gtk_widget_show (dialog_window);
}
```

CancelFunction 函数完成删除对话框和清除全局变量的工作。这样，可以再次创建对话框。

```
/*
 * CancelFunction
 *
 * Close the find dialog window.
 */
void CancelFunction (GtkWidget *widget, gpointer data)
{
    /* --- Close it. --- */
    gtk_widget_destroy (GTK_WIDGET (data));

    dialog_window = NULL;
}
```

可以用一种更好的方法代替每次用到时才去建立对话框然后再删除它。你可以只创建一次，然后在使用它的时候再使它成为可见。当用户完成在对话框中的工作后就将对话框隐藏，这样从表面上看它是被关掉了。虽然这种方法需要多用一些内存，但由于不需要再创建对话框而提高了显示速度。

FindFunction 函数在文件中执行查找字符串的工作。函数从当前光标位置开始进行查找。代替使用 gtk_test_get_point 函数找光标位置，在构件中使用了 selection_end_pos。这样做的理由是：当完成查找时，被找到的结果将被加亮。而在文中查找下一个值时需要从本次选择区域的后面开始。在你找到字符串后，你需要使用一点技巧才能使构件将加亮的部份滚动到合适的位置。（加亮正文的操作实际上并不滚动构件中的文本）。幸运的是，在构件中插入字符可以将文本滚动到插入点。但是，你仅仅在查找字符串。所以，你可以在正文滚动到插入点之后再将插入的字符删除。

```
/*
 * FindFunction
 *
 * Locks for the string in the notepad
 */
void FindFunction (GtkWidget *widget, gpointer data)
{
```

```
int nIndex;
GtkWidget *text = GetTextWidget ();
char *szHaystack;

/* --- Get the text from the widget --- */
szHaystack = GetText ();

/* --- Free up the old needle (text) --- */
if (szNeedle) {
    g_free (szNeedle);
}

/* --- Get text to search for. --- */
szNeedle = gtk_editable_get_chars (
              GTK_EDITABLE (entry), 0, -1);

/* --- Get cursor position --- */
nIndex = GTK_EDITABLE (text)->selection_end_pos;

/* --- Find the string --- */
nIndex = LookForString (szHaystack, szNeedle, nIndex);

if (nIndex >= 0) {

    /* --- Move the cursor to a position --- */
    gtk_text_set_point (GTK_TEXT (text), nIndex);

    /* --- These two lines will force the widget to --- */
    /*      scroll to the position where the text is. --- */
    gtk_text_insert (GTK_TEXT (text), NULL, NULL, NULL, "", 1);
    gtk_text_backward_delete (GTK_TEXT (text), 1);

    /* --- Select the field found. --- */
    gtk_editable_select_region (GTK_EDITABLE (text),
                                nIndex, nIndex + strlen (szNeedle));

    /* --- Allow it to he created again --- */
    dialog_window = NULL;
} else {

    ShowMessage ("Find...","Not found.  Reached the end of the file.");
}
```

```
    /* --- Release memory. --- */
    g_free (szHaystack);
}
```

别忘了你用 GetText 函数得到的字符串是需要释放的。因为它调用了 gtk_editable_get_chars 函数，这个函数为返回字符串分配了内存块。LookForString 函数从传入的位置开始进行查找，并返回得到的位置的索引。如果在正文构件中没有找到字符串则返回 -1 值。

```
/*
 * LookForString
 *
 * Search for a string (szNeedle) in a larger
 * string (szHaystack) starting at a given position (nStart)
 * in the large string.
 *
 * This is referred to as the brute force algorithm.
 */
int LookForString (char *szHaystack, char *szNeedle, int nStart)
{
    int nHaystackLength;
    int nNeedleLength;
    int nPos;

    /* --- Get the length of the strings. --- */
    nHaystackLength = strlen (szHaystack);
    nNeedleLength = strlen (szNeedle);

    /* --- Check each string --- */
    for (nPos = nStart; nPos < nHaystackLength; nPos++) {

        /* --- Did we find it here? --- */
        if (strncmp (&szHaystack[nPos], szNeedle, nNeedleLength) == 0) {

            /* --- yes, return index --- */
            return (nPos);
        }
    }

    /* Didn't find string, return -1 --- */
    return (-1);
}
```

8.7.6 进度条

进度条函数来自第 6 章“更多的构件”。你可以用进度条给用户提供信息，告诉用户那些需要长时间才能完成的操作的进度。当然，这里在装载文件时有意地设了一个小花样，在每次循环中使用小的缓冲区以降低装载速度。如果装载太快，进度条就没机会显示了。你可以改变缓冲的大小以加速文件的装载，这很容易做到。

8.8 小结

你可以使用 GTK + 快速创建应用。在本章中，GTK + 的构件在开发编辑器的过程中发挥了很大的作用。通过将菜单、工具条和编辑构件结合在一起，就可以创建简单的编辑应用程序。

第9章 扫　雷

扫雷是一个简单的游戏，因为微软将它放入视窗中而变得很流行。游戏的玩法是将盘面上所有没有藏雷的小方块翻开。虽然开始的几次选择具有很大的偶然性，但已翻开的小方块会给你提供一些线索，帮助你翻开其余的方块而不踩到雷。每一个方块都"知道"有几颗雷在它周围的方块中，所以每多翻开一个没有雷的方块都会有多一条线索来帮助你发现旁边藏雷的方块。当然，为了使它更加有趣，游戏还增加了一个记录所用时间的计时器，这样当你扫清了盘面上的所有雷后，创造最短时间记录将具有挑战性。图9.1显示的是扫雷场的游戏画面。

扫雷程序再次使用到了前面提到的大部份内容。事实上，扫雷应用和计算器应用有很多相似之处。它们都用到了组合表中的按钮构件，但扫雷程序还是要复杂一些，它增加了选择难度级别的菜单(包括初级、中级和高级)以及一个用来显示记时器和尚未找到的地雷数的工具条。此外，程序还使用了 xpm 图像，如程序中的地雷、笑脸按钮、彩色的数字和标雷的小旗子。

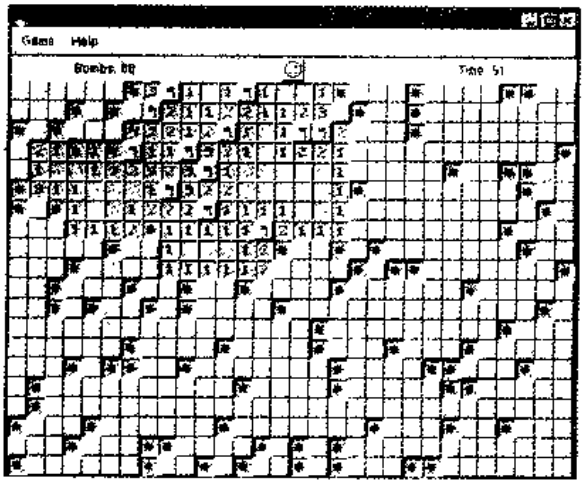


图 9.1　扫雷应用

所有按钮都存储在一个两维的 typMinesweeperButton 结构数组里面，在这个结构中保存着按钮构件信息，如：这个按钮下面有没有地雷、它周围有多少个按钮下面有地雷等。

```
/*
 * --- data structure to keep track of the Minesweeper buttons.
 */
typedef struct {
    int         buttonState;  /* --- Column to place the button --- */
    GtkWidget  *widget;       /* --- Handle to the button --- */
    int         nBombsNearby; /* --- How many near this cell? --- */
```

```
    int     bHasBomb;     /* --- Does it have a bomb? --- */
    int     nRow;         /* --- Row of button --- */
    int     nCol;         /* --- Column of button --- */
} typMinesweeperButton;
```

每次用户开始游戏时,计算机随机地将一定数量的地雷放在部份按钮下面,它还要将和每个按钮相邻的雷数算出来。不过用户一开始是看不到这些信息的,只有当用户点击了没有雷的按钮后,它才会被显示出来。

第一步是创建图像数据,这些图都是很简单和直观的。其中包括一些用于按钮的笑脸图案、还有标雷的小旗子、地雷(用户误踩的时候用)和一系列显示周围雷数的彩色数字。

9.1 bitmaps.h

扫雷游戏中使用的图像保存在 bitmaps.h 文件中。但是,存放彩色数字的位图不包括在内。注意这里的笑脸是由三个不同的图组成的——一个表示游戏在进行,一个表示你失败了,还有一个用于庆祝你的胜利。这三个图形并不需要太多的代码,但却给程序增添了几分特色。

```
/*
 * --- Flag for marking the bombs.
 */
static char  *xpm_flag[] = {
"12 12 4 1",
"  c None",
"X c #000000",
"R c #FF0000",
"r c #AA0000",
"            ",
"  RRRRRRR   ",
"  RRRRRrr   ",
"  RRRrrrr   ",
"  Rrrrrrr   ",
"        X   ",
"        X   ",
"        X   ",
"        X   ",
"        X   ",
"       XXX  ",
"            ",
};
```

```
/*
 * ---A bomb.  Oops, you're not as smart as you thought.
 */
static char *xpm_bomb[] = {
"12 12 4 1",
"  c None",
"X c #000000",
"R c #FF0000",
"r c #AA0000",
"            ",
"     X      ",
"  X  X  X   ",
"   XXXXX    ",
"   XXXXX    ",
" XXXXXXXXX  ",
"   XXXXX    ",
"   XXXXX    ",
"  X  X  X   ",
"     X      ",
"            ",
"            ",
};


/*
 * ---Wrong move!
 */
static char *xpm_bigx[] = {
"12 12 4 1",
"  c None",
"X c #000000",
"R c #FF0000",
"r c #AA0000",
"RRR      RRR",
" RRR    RRR ",
"  RRR  RRR  ",
"   RRRRRR   ",
"    RRRR    ",
"    RRRR    ",
"    RRRR    ",
"   RRRRRR   ",
"  RRR  RRR  ",
```

```
" RRR     RRR ",
"rRR       RRR",
"               ",
};


/*
 * --- Bitmap of a smile
 */
static char *xpm_smile[] = {
"16 16 4 1",
"  c None",
". c #000000",
"X c #FFFF00",
"r c #AA0000",
"      ......     ",
"    ..XXXXXX..   ",
" ..XXXXXXXXXX.   ",
" .XXXXXXXXXXXX. ",
" .XX..XXXX..XX. ",
".XXX..XXXX..XXX.",
".XXXXXXXXXXXXXX.",
".XXXXXXXXXXXXXX.",
".XXXXXXXXXXXXXX.",
".XXXXXXXXXXXXXX.",
" .XX.XXXXXX.XX. ",
" .XXX......XXX. ",
"   .XXXXXXXXXX.  ",
"    ..XXXXXX..   ",
"      ......     ",
"                 ",
};


/*
 * --- frown.  You lost.
 */
static char *xpm_frown[] = {
"16 16 4 1",
"  c None",
". c #000000",
"X c #FFFF00",
```

```
"r c #AA0000",
"      ......      ",
"    ..XXXXXX..    ",
"  ..XXXXXXXXXX.   ",
" .XXXXXXXXXXXX.   ",
" .XX.X.XX.X.XX.   ",
".XXXX.XXXX.XXXX.",
".XXX.X.XX.X.XXX.",
".XXXXXXXXXXXXXX.",
".XXXXXXXXXXXXXX.",
".XXXXXXXXXXXXXX.",
" .XXX......XXX.   ",
" .XX.XXXXXX.XX.   ",
"  .XXXXXXXXXX.    ",
"    ..XXXXXX..    ",
"      ......      ",
"                  ",
};


/*
 * ---We have a winner.
 */
static char *xpm_winner[] = {
"16 16 4 1",
"  c None",
". c #000000",
"X c #FFFF00",
"r c #AA0000",
"      ......      ",
"    ..XXXXXX..    ",
"  ..XXXXXXXXXX.   ",
" .XXXXXXXXXXXX.   ",
" .XX...XX...XX.   ",
".XX.........XX.",
".X.X...XX...X.X.",
"..XXXXXXXXXXXX..",
".XXXXXXXXXXXXXX.",
".XXXXXXXXXXXXXX.",
" .XX.XXXXXX.XX.   ",
" .XXX......XXX.   ",
"  .XXXXXXXXXX.    ",
```

```
"   ..XXXXXX..   ",
"   .  ......    ",
"                ",
};
```

9.2 Digits.h

显示方块周围雷数的彩色数字是像元图。因为所有的按钮数据都用像元图表示,所以在按钮上贴地雷、旗子或是数字可以调用同一个子程序——只是贴的图不一样。

```
/*
 * 1 - light blue
 * 2 - green
 * 3 - red
 * 4 - dark blue
 * 5 - brownish purple
 */

static const char *xpm_one[] = {
"12 12 2 1",
"  c None",
"X c #3333CC",
"            ",
"     XX     ",
"    XXX     ",
"   X XX     ",
"     XX     ",
"     XX     ",
"     XX     ",
"     XX     ",
"     XX     ",
"   XXXXXX   ",
"            ",
"            ",
};


static const char *xpm_two[] = {
"12 12 2 1",
"  c None",
```

```
"X c #009900",
"              ",
"   XXXXXX     ",
"  X      X    ",
"         XX   ",
"        XX    ",
"       XX     ",
"      XX      ",
"     XX       ",
"    XX        ",
"  XXXXXXXX    ",
"              ",
"              ",
};


static const char *xpm_three[] = {
"12 12 2 1",
"  c None",
"x c #AA0000",
"              ",
"   XXXXX      ",
"        XX    ",
"        XX    ",
"   XXXXX      ",
"        XX    ",
"        XX    ",
"        XX    ",
"        XX    ",
"  XXXXXX      ",
"              ",
"              ",
};


static const char *xpm_four[] = {
"12 12 2 1",
"  c None",
"X c #000066",
"              ",
"  XX    XX    ",
"  XX    XX    ",
```

```
"  XX     XX  ",
"  XX     XX  ",
"  XXXXXXXX  ",
"         XX  ",
"         XX  ",
"         XX  ",
"         XX  ",
"             ",
"             ",
};


static const char * xpm_five[] = {
"12 12 2 1",
"  c None",
"X c #992299",
"             ",
"  XXXXXXX  ",
"  XX       ",
"  XX       ",
"  XXXXXX   ",
"         XX  ",
"         XX  ",
"         XX  ",
"  XX     XX  ",
"  XXXXXX   ",
"             ",
"             ",
};


static const char * xpm_six[] = {
"12 12 2 1",
"  c None",
"X c #550055",
"             ",
"   XXXXX   ",
"  XX       ",
"  XX       ",
"  XXXXXX   ",
"  XX     XX  ",
```

```
"  XX    XX  ",
"  XX    XX  ",
"  XX    XX  ",
"   XXXXXX   ",
"            ",
"            ",
};


static const char * xpm_seven[] = {
"12 12 2 1",
".  c None",
"X c  # 550000",
"            ",
"  XXXXXXX   ",
"        XX  ",
"       XX   ",
"       XX   ",
"      XX    ",
"      XX    ",
"     XX     ",
"     XX     ",
"     XX     ",
"            ",
"            ",
};


static const char * xpm_eight[] = {
"12 12 2 1",
"  c None",
"X c  # 44:144",
"            ",
"   XXXXXX   ",
"  XX    XX  ",
"  XX    XX  ",
"   XXXXXX   ",
"  XX    XX  ",
"  XX    XX  ",
"  XX    XX  ",
```

```
"  XX    XX  ",
"   XXXXXX   ",
"            ",
"            ",
};
```

9.3 Timer.c

游戏需要跟踪并不断刷新显示的时间的秒数。时间是从用户第一次在盘面上点击按钮时算起的。用户第一次点击按钮时调用函数 StartTimer。实际上每次用户点击按钮时这个函数都会被调用,但如果计时器已经运作,StartTimer 就会自动忽略这些调用。StartTimer 函数启动一个回呼函数(TimerCallback),它每一秒钟都要被调用一次来刷新和显示时间。如果用户误踩到了雷或是玩赢了、或是单击了笑脸键重新开始游戏,StopTimer 函数就会被调用。

```
/*
 * Timer.c
 *
 * Auth: Eric Harlow
 *
 * Routines to update something every second.
 */

#include <gtk/gtk.h>


static int nSeconds = 0;
static gint timer = 0;
static int bTimerRunning = FALSE;

void UpdateSeconds (int);

/*
 * TimerCallback
 *
 * Every second, this will be called to update the time
 * in the clock window.
 */
gint TimerCallback (gpointer data)
{
    /* --- Another second has gone by --- */
    nSeconds++;
```

```
    UpdateSeconds (nSeconds);
}


/*
 * StartTimer
 *
 * Starts up the time.  Happens when the user first
 * clicks on a button.
 */
void StartTimer ()
{

    /* --- If the timer isn't already running --- */
    if (!bTimerRunning) {

        /* --- Start at zero --- */
        nSeconds = 0;

        /* --- Call function 'timerCallback' every 1000ms --- */
        timer = gtk_timeout_add (1000, TimerCallback, NULL);

        /* --- Timer is running. --- */
        bTimerRunning = TRUE;
    }
}


/*
 * StopTimer
 *
 * Stops the timer.  User probably hit a bomb.
 */
void StopTimer ()
{
    /* --- If the time is running. --- */
    if (bTimerRunning) {

        /* --- Stop the timer. --- */
        gtk_timeout_remove (timer);

        /* --- Fix the flag. --- */
```

```
        bTimerRunning = FALSE;
    }
}
```

9.4 Minesweeper.c

游戏本身是一个 N × M 的触发按钮网格表。按钮是用 typMinesWeeperButton 数据结构表示的。每个按钮在网格中都有一个行(nRow)和一个列(nCol)位置。同时,它还有一个标志用来表示其下有没有藏雷(bHasBomb)。除此之外,还有一个构件变量和一个计数器。其中变量在建立触发按钮时用来保存生成的按钮指针,而计数器则用来记录相邻方块下的雷数(只有在本按钮下没有雷的情况下才会用到这个值)。buttonState 反映按钮当前所处的状态。按钮最初被设成 BUTTON _ UNKNOWN 状态,但用户可以通过单击鼠标右键将状态改为 BUTTON _ FLAGGED,这时按钮上也会增加一面小旗子以提示用户。BUTTON _ DOWN 状态表示按钮被按下,如果这个按钮下有雷,那么此状态就会导致游戏结束。最后还有一种状态要讲,那就是 BUTTON _ QUESTION,用户可以用它标出那些有疑问的按钮,这时按钮上出现的是一个问号,表示本按钮下面可能有雷。这种状态很容易加,但在本程序中并没有使用,读者可以拿它当作练习自己去试试。

循环递归可以用来简化程序。例如,如果用户点击了一个没有雷的按钮,而这个按钮周围也没有雷,程序就会继续翻开其周围的方块就好象用户已经点击了它们。假设其中又有周围的雷数为零的按钮,那么它周围的按钮也同样被翻开,如此循环下去。

当按钮被创建时,点击它们时所使用的回呼函数只有一个。用按钮在 typMinesweeperButton 数组中的数据作为回呼函数的参数。虽然回呼函数的参数只和被点击的按钮有关,但通过此按钮的行值和列值,我们仍可以找到表中和这个按钮相邻的所有按钮。

```
/*
 * File: minesweeper.c
 * Auth: Eric Harlow
 *
 * Similar to the Windows Minesweeper game.
 */

#include <gtk/gtk.h>
#include <stdlib.h>
#include <time.h>
#include "misc.h"
#include "bitmaps.h"
#include "digits.h"

void SetStartButtonIcon (gchar **xpm_data);
```

```
/*
 * --- Want the buttons to be a particular size.
 */
#define BUTTON_WIDTH 18
#define BUTTON_HEIGHT 19


/*
 * --- Each of the digits showing the number of bombs
 *     has a colorful bitmap showing the count.  This
 *     array makes the looking up of the digits happen
 *     very quickly.
 */
gpointer digits[] = {
    NULL,
    xpm_one,
    xpm_two,
    xpm_three,
    xpm_four,
    xpm_five,
    xpm_six,
    xpm_seven,
    xpm_eight
};


/*
 * --- These are the available button states.
 *
 * Unknown - Empty.  Don't know what it could be.
 * Flagged - User has put a flag on it, suspecting
 *           there's a bomb underneath.
 * Question - not implemented.
 * Down - Button has been pressed.
 */
enum {
    BUTTON_UNKNOWN,
    BUTTON_FLAGGED,
    BUTTON_QUESTION,
    BUTTON_DOWN
};
```

```
/*
 * --- data structure to keep track of the minesweeper buttons.
 */
typedef struct {
    int         buttonState;  /* --- Column to place the button --- */
    GtkWidget  *widget;       /* --- Handle to the button --- */
    int         nBombsNearby; /* --- How many are near this cell? --- */
    int         bHasBomb;     /* --- Does it have a bomb? --- */
    int         nRow;         /* --- Row of button --- */
    int         nCol;         /* --- Column of button --- */
} typMinesweeperButton;

/*
 * --- Default values for the size of the grid
 *      and number of bombs.
 */
static int nRows = 10;
static int nCols = 10;
static int nTotalBombs = 10;


/*
 * --- Use a 2-d array for the grid.  This is just
 *      the maximum size.
 */
#define MAX_ROWS 35
#define MAX_COLS 35
/*
 * Globals.
 */

/* --- Just allocate the MAX for now.  This really should be dynamic,
 *      but using a two-dimensional array is easy when dealing with a grid.
 */
typMinesweeperButton mines[MAX_COLS][MAX_ROWS];

/* --- Flags for the game --- */
int bGameOver = FALSE;              /* --- Game is over? --- */
int bResetGame = FALSE;             /* --- Game is over? --- */
int nBombsLeft;                     /* --- Undiscovered bombs --- */

GtkWidget *table = NULL;            /* --- Table with the grid in it. */
```

```
GtkWidget *button_start;          /* --- Start button --- */
GtkWidget *label_bombs;           /* --- Label showing # of bombs left */
GtkWidget *label_time;            /* --- Label showing # of bombs left */
GtkWidget *vbox;                  /* --- main window's vbox --- */

/*
 * --- Prototypes.
 */
void DisplayHiddenInfo (typMinesweeperButton *mine);
void CreateMinesweeperButtons (GtkWidget *table, int c, int r, int flag);
void FreeChildCallback (GtkWidget *widget);



/*
 * CheckForWin
 *
 * Check to see if the game has been won.  Game has been won when the
 * number of unclicked squares equals the number of total bombs.
 */
void CheckForWin ()
{
    int i, j;
    int nMines = 0;

    /* --- Run through all the squares --- */
    for (i = 0; i <  nCols; i++) {
        for (j = 0; j < nRows; j++) {

            /* --- If square is unclicked or has a flag on it... --- */
            if (mines[i][j].buttonState == BUTTON_UNKNOWN ||
                mines[i][j].buttonState == BUTTON_FLAGGED) {

                /* --- Could be a bomb. --- */
                nMines++;
            }
        }
    }

    /* --- As many bombs as there are squares left? --- */
    if (nMines == nTotalBombs) {

        /* --- Stop the game.  We have a winner. --- */
```

```
        StopTimer ();
        SetStartButtonIcon ((gchar ** ) xpm_winner);
        bGameOver = TRUE;
    }
}


/*
 * AddImageToMine
 *
 * Change the image on the button to be that of the xpm.
 *
 * mine - square on the grid
 * xpm - image data
 */
void AddImageToMine (typMinesweeperButton * mine, char * xpm[])
{
    GtkWidget * widget;

    /* --- Create a widget from the xpm file --- */
    widget = CreateWidgetFromXpm (table, (gchar ** ) xpm);

    /* --- Put the bitmap in the button --- */
    gtk_container_add (GTK_CONTAINER (mine->widget), widget);

    /* --- deref image so when button is destroyed, image is destroyed. */
    gdk_pixmap_unref ((GdkPixmap * ) widget);
}


/*
 * UpdateSeconds
 *
 * Refresh the seconds display in the toolbar.
 *
 * nSeconds - how many seconds to display.
 */
void UpdateSeconds (int nSeconds)
{
    char buffer[44];

    /* --- Change the label to show new time --- */
```

```
    sprintf (buffer, "Time: %d", nSeconds);
    gtk_label_set (GTK_LABEL (label_time), buffer);
}


/*
 * DisplayBombCount
 *
 * Show number of bombs remaining.
 */
void DisplayBombCount ()
{
    char buffer[33];

    /* --- You have XX bombs left --- */
    sprintf (buffer, "Bombs: %d", nBombsLeft);
    gtk_label_set (GTK_LABEL (label_bombs), buffer);
}


/*
 * FreeChild
 *
 * Free all the children of the widget
 * This is called when the button has to display a new image.
 * The old image is destroyed here.
 */
void FreeChild (GtkWidget *widget)
{
    /* --- Free button childrer. --- */
    gtk_container_foreach (
            GTK_CONTAINER (widget),
            (GtkCallback) FreeChildCallback,
            NULL);

}

/*
 * delete_event
 *
 * The window is closing down, end the gtk loop
 *
```

```
 */
void delete_event (GtkWidget *widget, gpointer *data)
{
    gtk_main_quit ();
}


/*
 * ShowBombs
 *
 * They clicked on a bomb, so now we have to show them where the
 * bombs really are.  (At least those they didn't find already.)
 * We display the bombs they didn't find as well as the bombs
 * they think they found that were not bombs.
 */
void ShowBombs (typMinesweeperButton *minefound)
{
    int i, j;
    typMinesweeperButton *mine;
    GtkWidget *widget_x;

    /* --- Run through all the squares --- */
    for (i = 0; i <  nCols; i++) {
        for (j = 0; j < nRows; j++) {

            /* --- Get the datastructure --- */
            mine = &mines[i][j];

            /* --- If there's a button here and there's a bomb --- */
            /* --- underneath --- */
            if (mine->buttonState == BUTTON_UNKNOWN &&
                mine->bHasBomb) {

                /* --- Display the bomb. --- */
                DisplayHiddenInfo (mine);

            /* --- If they marked it as a bomb and there is
             *      no bomb here... --- */
            } else if (mine->buttonState == BUTTON_FLAGGED &&
                       ! mine->bHasBomb) {

                /* --- Free the flag --- */
```

```
                FreeChild (mine->widget);

                /* --- Show the X at the location --- */
                AddImageToMine (mine, xpm_bigx);
            }
        }
    }
}


/*
 * OpenNearbySquares
 *
 * Open up all squares around this square.
 *
 * col, row - position to open up the square
 * Open all squares near this one - X represents
 * the current square.
 *
 *      |---|---|---
 *      |   |   |   |
 *      ----------
 *      |   | X |   |
 *      ----------
 *      |   |   |   |
 *      |---|---|---
 */
void OpenNearbySquares (int col, int row)
{
    int i, j;

    /* --- Look one column back and one column ahead --- */
    for (i = MAX (col-1, 0); i <= MIN (col+1, nCols-1); i++) {

        /* --- Check one row back and one row ahead --- */
        for (j = MAX (row-1, 0); j <= MIN (row+1, nRows-1); j++) {

            /* --- Display what's underneath --- */
            DisplayHiddenInfo (&mines[i][j]);
        }
    }
```

```
}


/*
 * DisplayHiddenInfo
 *
 * Display what's hidden underneath the button.
 * Could be a bomb -
 * Could be a square with a count of the bombs
 *   nearby.
 * Could be an empty square
 */
void DisplayHiddenInfo (typMinesweeperButton *mine)
{
    char       buffer[88];
    GtkWidget *widget;

    /* --- If it's already down, just return --- */
    if (mine->buttonState == BUTTON_DOWN) {
        gtk_toggle_button_set_state (GTK_TOGGLE_BUTTON (mine->widget), TRUE);
        return;
    }

    /* --- If the button is flagged, don't fix it for them --- */
    if (mine->buttonState == BUTTON_FLAGGED) {

        /* --- They said there's a bomb here - so don't
         *      open it up even if logically, there
         *      can't be a bomb.
         */
        gtk_toggle_button_set_state (GTK_TOGGLE_BUTTON (mine->widget), FALSE);

    } else {

        /* --- Put the button in the "down" state --- */
        mine->buttonState = BUTTON_DOWN;
        gtk_toggle_button_set_state (GTK_TOGGLE_BUTTON (mine->widget), TRUE);

        /* --- If there's a bomb at the location --- */
        if (mine->bHasBomb) {

            /* --- Show the bomb at the location --- */
```

```
            AddImageToMine (mine, xpm_bomb);

        /* ---No bombs, but there are bombs near this one--- */
        } else if (mine->nBombsNearby) {

            /* --- Show the count of nearby bombs. --- */
            AddImageToMine (mine, digits[mine->nBombsNearby]);

        } else {

            /* --- Hmm.  Clicked here, but no bombs and no count. --- */
            /* --- Open up all squares near here - may cascade. --- */
            OpenNearbySquares (mine->nCol, mine->nRow);
        }
    }
}


/*
 * ResetGame
 *
 * Reset the game so it can be replayed.  Reset bomb
 * count and create a nice empty field of bombs.
 */
void ResetGame (int nGridColumns, int nGridRows, int nBombs, int bNewButtons)
{

    /* --- Reset the number of bombs in grid  --- */
    nTotalBombs = nBombs;

    /* --- Reset the number of bombs undiscovered --- */
    nBombsLeft = nBombs;

    /* --- Create the minesweeper buttons. --- */
    CreateMinesweeperButtons (table, nGridColumns, nGridRows, bNewButtons);

    /* --- Stop the timer. --- */
    StopTimer ();

    UpdateSeconds (0);

    SetStartButtonIcon ((gchar **) xpm_smile);
```

```
}

/*
 * FreeChildCallback
 *
 * Free the widget.
 */
void FreeChildCallback (GtkWidget *widget)
{
    gtk_widget_destroy (widget);
}


/*
 * SetStartButtonIcon
 *
 * Set the start button to have the image based on
 * the data passed in.  Usually, this is going to be
 * either the happy face or the frown.
 */
void SetStartButtonIcon (gchar **xpm_data)
{
    GtkWidget *widget;

    /* --- Create a widget from the xpm --- */
    widget = CreateWidgetFromXpm (button_start, xpm_data);

    /* --- Free any children the button has --- */
    FreeChild (button_start);

    /* --- Make this the current image --- */
    gtk_container_add (GTK_CONTAINER (button_start), widget);
}


/*
 * start_button_clicked
 *
 * Event handler for the clicking of the start
 * button
 */
void start_button_clicked (GtkWidget *widget, gpointer *data)
```

```
    SetStartButtonIcon ((gchar * * ) xpm_smile);
    ResetGame (nCols, nRows, nTotalBombs, FALSE);
}


/*
 * mine_button_clicked
 *
 * Event handler for one of the mine buttons being
 * clicked.
 *
 * widget - button pressed.
 * data - button label.
 */
void mine_button_clicked (GtkWidget * widget, gpointer * data)
{
    typMinesweeperButton * mine;
    GtkWidget * label;

    mine = (typMinesweeperButton * ) data;

    printf ("button clicked - game over = %d \n", bGameOver);

    /* --- If the game is over --- */
    if (bGameOver) {
        /* --- Leave the button as it was. --- */
        gtk_toggle_button_set_state (GTK_TOGGLE_BUTTON (widget),
                                     (mine-> buttonState == BUTTON_DOWN));
        return;
    }

    /* --- If game is being reset --- */
    if (bResetGame) return;

    /* --- Start the time ... now --- */
    StartTimer ();

    /* --- If they clicked on a mine button... --- */
    if (mine-> bHasBomb) {

        /* --- Game over, dude! --- */
```

```
        bGameOver = TRUE;

        /* --- Smiley face is frowning. --- */
        SetStartButtonIcon((gchar **) xpm_frown);

        /* --- Display all unseen bombs. --- */
        StopTimer();
        ShowBombs(mine);

    } else {

        /* --- Create a label for the cell and put the --- */
        /* --- number of nearby bombs in it. --- */
        DisplayHiddenInto(mine);
        CheckForWin();
    }
}


/*
 * button_press
 *
 * The button press could be a right mouse button
 * that needs to be handled by rotating the
 * image in the button.
 */
void button_press(GtkWidget *widget, GdkEventButton *event, gpointer *data)
{
    typMinesweeperButton *mine;
    GtkWidget *pixmapWidget;

    /* --- Ignore if game is already over--- */
    if(bGameOver) {
        return;
    }

    /* --- Which mine was clicked? --- */
    mine = (typMinesweeperButton *) data;

    /* --- Make sure it's a button event --- */
    if(event->type == GDK_BUTTON_PRESS) {
```

```
        /* --- Was it the right mouse button? --- */
        if (event->button == 3) {

            switch (mine->buttonState) {

                case BUTTON_UNKNOWN:

                    /* --- Free button children --- */
                    FreeChild (widget);

                    mine->buttonState = BUTTON_FLAGGED;
                    AddImageToMine (mine, xpm_flag);
                    nBombsLeft --;
                    break;

                case BUTTON_FLAGGED:

                    /* --- Free button children --- */
                    FreeChild (widget);

                    /* --- Free button children --- */
                    mine->buttonState = BUTTON_UNKNOWN;
                    nBombsLeft ++;
                    break;
            }
            DisplayBombCount ();
            CheckForWin ();
        }
    }
}


/*
 * CreateButton
 *
 * Create a button, assign event handlers, and attach the button to the
 * table in the proper place.
 */
GtkWidget *CreateButton (GtkWidget *table,
                         typMinesweeperButton *mine,
                         int row,
                         int column)
```

```
{
    GtkWidget *button;

    /* --- Create the button --- */
    button = gtk_toggle_button_new();

    /* --- Init the button fields --- */
    mine->buttonState = BUTTON_UNKNOWN;
    mine->nRow = row;
    mine->nCol = column;

    /* --- We care if the button is clicked --- */
    gtk_signal_connect(GTK_OBJECT(button), "clicked",
                       GTK_SIGNAL_FUNC(mine_button_clicked), mine);

    /* --- We care about other mouse events. --- */
    gtk_signal_connect(GTK_OBJECT(button), "button_press_event",
                       GTK_SIGNAL_FUNC(button_press), mine);

    /* --- Put the button in the table in the right place. --- */
    gtk_table_attach(GTK_TABLE(table), button,
                     column, column+1,
                     row + 1, row + 2,
                     GTK_FILL | GTK_EXPAND,
                     GTK_FILL | GTK_EXPAND,
                     0, 0);

    /* --- Set the button to a uniform size --- */
    gtk_widget_set_usize(button, BUTTON_WIDTH, BUTTON_HEIGHT);

    /* --- Make the button visible --- */
    gtk_widget_show(button);

    /* --- return the button. --- */
    return(button);
}



/*
 * CountNearbyBombs
 *
```

```
 * Count the number of bombs this square is next to.
 */
int CountNearbyBombs (int col, int row)
{
    int i, j;
    int nCount = 0;

    /* --- Every square that would be at most 1 square away --- */
    for (i = MAX (col-1, 0); i <= MIN (col + 1, nCols-1); i++) {
        for (j = MAX (row-1, 0); j <= MIN (row + 1, nRows-1); j++) {

            /* --- If it's got a bomb --- */
            if (mines[i][j].bHasBomb) {

                /* --- Keep track of the count. --- */
                nCount++;
            }
        }
    }
    return (nCount);
}


/*
 * CreateMinesweeperButtons
 *
 * Create the buttons on the minesweeper from the table we defined at the
 * beginning of this program.  The button pointers (handles) are stored
 * back in the table so they can be referenced later.
 */
void CreateMinesweeperButtons (GtkWidget *table,
                               int nGridColumns,
                               int nGridRows,
                               int bNewButtons)
{
    int ci;
    int ri;
    GtkWidget *button;
    int nBombs;
    typMinesweeperButton *mine;

    /* --- Update the global variables --- */
```

```
nCols = nGridColumns;
nRows = nGridRows;

bGameOver = FALSE;
bResetGame = TRUE;

/* --- Update bomb count. --- */
DisplayBombCount ();

/* --- Check each button --- */
for (ci = 0; ci < nCols; ci++) {
    for (ri = 0; ri < nRows; ri++) {

        /* --- The button has nothing at all. --- */
        mine = &mines[ci][ri];
        mine->bHasBomb = 0;
        mine->buttonState = BUTTON_UNKNOWN;

        /* --- Widget assoc to the mine? --- */
        if (bNewButtons) {

            /* --- Create a button --- */
            mine->widget = CreateButton (table, mine, ri, ci);
        } else {

            /* --- Reuse button --- */

            /* --- Free any existing xpm/label --- */
            FreeChild (mine->widget);

            /* --- Put button up. --- */
            gtk_toggle_button_set_state (
                        GTK_TOGGLE_BUTTON (mine->widget),
                        FALSE);
        }
    }
}

/* --- Place the bombs. --- */
nBombs = nTotalBombs;

/* --- While we have bombs to deliver --- */
```

```
    while (nBombs > 0) {

        /* --- Calculate a row/col position --- */
        ci = rand() % nCols;
        ri = rand() % nRows;

        /* --- If no bomb exists, create one! --- */
        if (mines[ci][ri].bHasBomb == 0) {
            mines[ci][ri].bHasBomb = 1;
            nBombs--;
        }
    }

    /* --- Once all bombs have been distributed, calculate
     *      how many bombs are adjacent to each button.
     */

    /* --- Check every button --- */
    for (ci = 0; ci < nCols; ci++) {
        for (ri = 0; ri < nRows; ri++) {

            mine = &mines[ci][ri];

            /* --- How many buttons? --- */
            mine->nBombsNearby = CountNearbyBombs (ci, ri);
        }
    }
    bResetGame = FALSE;
}


/*
 * SetGrid
 *
 * Sets the game grid to the size specified with the
 * number of bombs.
 */
void SetGrid (int nGridColumns, int nGridRows, int nBombs)
{
    int row, col;

    /* --- If the packing table exists. --- */
```

```
    if (table) {

        /* --- Destroy it and all the buttons. --- */
        gtk_widget_destroy (table);
    }

    /* --- Create a table for the buttons --- */
    table = gtk_table_new (nGridColumns, nGridRows, FALSE);

    /* --- Add it to the vbox --- */
    gtk_box_pack_start (GTK_BOX (vbox), table, FALSE, FALSE, 0);

    /* --- Show the table. --- */
    gtk_widget_realize (table);

    /* --- Do a game reset with the numbers --- */
    ResetGame (nGridColumns, nGridRows, nBombs, TRUE);

    /* --- Show the table. --- */
    gtk_widget_show (table);
}



/*
 * main
 *
 * Program begins here
 */
int main (int argc, char *argv[])
{
    GtkWidget *window;
    GdkBitmap *mask;
    GtkStyle  *style;
    GtkWidget *widget_smile;
    GtkWidget *hbox;

    /* --- GTK initialization --- */
    gtk_init (&argc, &argv);

    /* --- Create the top window --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

```
/* --- Don't allow window to be resized. --- */
gtk_window_set_policy (GTK_WINDOW (window), FALSE, FALSE, TRUE);

/* --- Give the window a title. --- */
gtk_window_set_title (GTK_WINDOW (window), "minesweeper");

/* --- You should always remember to connect the destroy event
 *      to the main window.
 */
gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                    GTK_SIGNAL_FUNC (delete_event), NULL);

vbox = gtk_vbox_new (FALSE, 1);
gtk_widget_show (vbox);

/* --- Create the application menu --- */
CreateMenu (window, vbox);

/* --- Horizontal box for score/start button --- */
hbox = gtk_hbox_new (TRUE, 1);
gtk_widget_show (hbox);

gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 0);

/*
 * --- Bombs left on the page is a label
 */

/* --- Add label with # of bombs left. --- */
label_bombs = gtk_label_new (" ");
gtk_box_pack_start (GTK_BOX (hbox), label_bombs, FALSE, FALSE, 0);
gtk_widget_show (label_bombs);

/*
 * --- Create the start button with smilely face on it
 */
button_start = gtk_button_new ();

/* --- We care if the button is clicked --- */
gtk_signal_connect (GTK_OBJECT (button_start),
                    "clicked",
```

```
                        GTK_SIGNAL_FUNC (start_button_clicked),
                        NULL);

    gtk_box_pack_start (GTK_BOX (hbox), button_start, FALSE, FALSE, 0);
    gtk_widget_show (button_start);

    /*
     * --- Time on the right
     */

    /* --- Add label with # of bombs left --- */
    label_time = gtk_label_new ("");
    gtk_box_pack_start (GTK_BOX (hbox), label_time, FALSE, FALSE, 0);
    gtk_widget_show (label_time);

    /* --- Make them visible --- */
    gtk_widget_show (vbox);

    /* --- Add application vbox to main window --- */
    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show (window);

    /* --- Create the *smile* and put it on the start button --- */
    SetStartButtonIcon ((gchar **) xpm_smile);

    /* --- Create 10x10 grid. --- */
    SetGrid (10, 10, 10);

    /* --- Start gtk event processing --- */
    gtk_main ();
    exit (0);
}
```

9.5 Menu.c

Menu.c 文件中的函数用于菜单项的产生和选择。扫雷程序的界面保持为最小。

```
/*
 * Minesweeper GUI application front end.
 *
 * Auth: Eric Harlow
```

```
 *
 */

#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <gtk/gtk.h>
#include "misc.h"


void SetGrid (int nColumns, int nRows, int nBombs);


/*
 * --- Global variables
 */
GtkWidget           *win_main;
GtkAccelGroup       *accel_group;
GtkWidget           *toolbar;


/*
 * menu_New
 *
 * Called when the "new" menu is selected.
 * Creates a new game for the user.
 */
void menu_New (GtkWidget *widget, gpointer data)
{
    /* --- Parameters are not used... --- */

    /* --- Pretend the start button was clicked. --- */
    start_button_clicked (NULL, NULL);
}


/*
 * funcBeginner
 *
 * Picked the "beginner" option from the menu.
 * Create a small grid.
 */
void funcBeginner (GtkWidget *widget, gpointer data)
```

```
}
    if (GTK_CHECK_MENU_ITEM(widget)->active) {
        SetGrid(10, 10, 10);
    }
}

/*
 * funcIntermediate
 *
 * Picked the "intermediate" option from the menu.
 * Creates a medium-sized grid.
 */
void funcIntermediate(GtkWidget *widget, gpointer data)
{
    if (GTK_CHECK_MENU_ITEM(widget)->active) {
        SetGrid(20, 15, 40);
    }
}

/*
 * funcAdvanced
 *
 * User picked the "advanced" option in the menu.
 * Make the largest grid with the most bombs.
 */
void funcAdvanced(GtkWidget *widget, gpointer data)

    /* --- If this item is now active --- */
    if (GTK_CHECK_MENU_ITEM(widget)->active) {

        /* --- Set the grid size --- */
        SetGrid(30, 20, 100);
    }
}

/*
 * menu_Quit
 *
 * Choose quit in the menu.
 * Exit the game.
 */
void menu_Quit(GtkWidget *widget, gpointer data)
```

```
}
    gtk_main_quit ();
}

/*
 * menu_About ()
 *
 * User chose the "about" menu item.
 * Show information about the application.
 */
void menu_About (GtkWidget *widget, gpointer data)
{

    ShowAbout ();
}

/*
 * CreateMainWindow
 *
 * Create the main window and the menu/toolbar associated with it
 */
void CreateMenu (GtkWidget *window, GtkWidget *vbox_main)
{
    GtkWidget *menubar;
    GtkWidget *menu;
    GtkWidget *menuitem;
    GSList *group = NULL;

    win_main = window;

    /* --- Create accel table --- */
    accel_group = gtk_accel_group_new ();
    gtk_accel_group_attach (accel_group, GTK_OBJECT (window));

    /* --- Menu Bar --- */
    menubar = gtk_menu_bar_new ();
    gtk_box_pack_start (GTK_BOX (vbox_main), menubar, FALSE, TRUE, 0);
    gtk_widget_show (menubar);

    /* --------------
       --- File menu ---
       -------------- */
```

```
    menu = CreateBarSubMenu (menubar, "Game");

    menuitem = CreateMenuItem (menu, "New","N",
                    "NAew Game",
                    GTK_SIGNAL_FUNC (menu_New), NULL);

    menuitem = CreateMenuItem (menu, NULL, NULL,
                    NULL, NULL, NULL);

    menuitem = CreateMenuRadio (menu, "Beginner", &group,
                    GTK_SIGNAL_FUNC (funcBeginner), NULL);

    menuitem = CreateMenuRadio (menu, "Intermediate", &group,
                    GTK_SIGNAL_FUNC (funcIntermediate), NULL);

    menuitem = CreateMenuRadio (menu, "Advanced", &group,
                    GTK_SIGNAL_FUNC (funcAdvanced), NULL);

    menuitem = CreateMenuItem (menu, NULL, NULL,
                    NULL, NULL, NULL);

    menuitem = CreateMenuItem (menu, "Quit", "",
                    "What's more descriptive than quit?",
                    GTK_SIGNAL_FUNC (menu_Quit), "quit");

    /* ------------
       --- Help menu ---
       ------------ */
    menu = CreateBarSubMenu (menubar, "Help");

    menuitem = CreateMenuItem (menu, "About Minesweeper", NULL,
                    "About the Minesweeper",
                    GTK_SIGNAL_FUNC (menu_About), "about");


}
```

9.6 其余的文件

还有一些文件是以前已经讲过的，它们可以从 www.mcp.com 下载（参考第 1 章，“介绍 GTK+”）。这些文件是用于菜单和帮助对话框的支持函数。对话框部份可以在第 6 章“更多

的构件”)中找到,支持菜单的函数则可以在第5章(“菜单、工具条和工具提示”)中找到。

9.7 小结

编写扫雷游戏能够进一步巩固你学到的知识:如何将工具条、菜单和其他构件放在一起来完成一项小的应用。对于可以用构件来实现的应用来说,GTK+库的使用是很灵活的。

第三部分　绘图、颜色和 GDK

第 10 章　绘图工具集
第 11 章　式样、颜色、字体、光标和引用
第 12 章　使用 GDK 的分子观察程序
第 13 章　可移动影像和动画制作

第 10 章　绘图工具集

绘图工具集(GDK,Graphics Drawing Kit)是在 GTK + 和特定操作系统的应用编程接口(API,Application Programming Interface)之间的低级绘制层——Linux 的 API 是 Xlib。由于 GTK 和机器的 API 之间没有直接接口,移植 GTK + 就成为移植 GLIB 和 GDK 的问题。GDK 提供像元的绘制能力以及低层的窗口建立和处理功能。对许多应用讲,使用构件是合适的;但是,如果你想用 GTK + 编写模拟时钟的应用程序,由于它没有绘制时钟表面的能力,实现这样的任务就有困难了。使用带 GDK 的绘图区(drawing area)构件代替封装的(canned)的构件,将使你能绘制任何你想绘制的内容。

10.1　绘图子程序

使用 GDK 子程序编写应用程序并不比直接使用 Xlib 好多少。幸运的是,GTK + 提供一种构件,这就是绘图区(drawing area)构件。你可以用它来建立需要手工绘图的应用。可以像使用其他 GTK + 构件一样使用绘图区构件。而且,它在建立依赖图形的应用时又足够灵活。这种方法的优点是可以在同一个应用中使用 GTK + 和 GDK。GTK + 提供菜单、工具条和其他构件,支持在绘图区构件内进行绘图。而 GDK 则提供用于绘制线、框、像元、圆和其他图形的 API。

每个 GDK 子程序至少取两个参数:可绘制区(GdkDrawable)和 GdkGC。GdkDrawable 表示可以在它上而进行绘图的区域。GdkGC 包含颜色和字体信息以及其他绘图信息。

10.1.1　画像元

GDK 提供在窗口和构件(如绘图区构件)内绘图的子程序。最简单的子程序是在可绘制区画单个像元的 gdk _ draw _ point 子程序。

```
/* --- Draw a point at 10, 10 --- */
gdk_draw_point (drawable, gc, 10, 10);
```

在本例及其他例子中,drawable 表示可以在其中绘图的区域。通常用 gc 表示颜色信息。在本章后面,将讨论如何取得和设置这些参数。

10.1.2　画直线

可以用 gdk _ draw _ line 函数在两点之间画一条直线。

```
/* --- Draw a diagonal line --- */
gdk_draw_line (drawable, gc, 0, 0, 10, 10);
```

10.1.3 画矩形

函数 gdk _ draw _ rectangle 在可绘制区内画一个框。可以填充这个框;这一技术常用来清除屏幕上需要重新绘制的区域。坐标中前两个数字表示框的左上角,后两个数字表示框的宽和高。第 3 个参数表示是否需要填充这个框,还是保持中空。TRUE 表示应该填充这个框。

```
/* --- Draw a box on the screen (0, 0), (20, 20) --- */
gdk_draw_rectangle (drawable, gc, FALSE, 0, 0, 20, 20);

/* --- Fill in a box on the screen (10, 10) - (30, 30) --- */
gdk_draw_rectangle (drawable, gc, TRUE, 10, 10, 20, 20);
```

10.1.4 画多角形

画奇特的形状(如头的外形或者 8 角形)需要画许多条线。通常不会多次重复使用 gdk _ draw _ line 函数,而是更方便地使用画多角形的函数 gdk _ draw _ polygon。这一函数使用待连接点的数组作为参数。如果数组的第一个点和最后一个点不同,将把它们连接起来。如果将填充参数设置为 TRUE,将填充多角形。

下面是画三角形的一种方法。

```
/* --- Top point of triangle --- */
points[0].x = 50;
points[0].y = 0;

/* --- Bottom-left side of triangle --- */
points[1].x = 0;
points[1].y = 50;

/* --- Bottom-right side of triangle --- */
points[2].x = 100;
points[2].y = 50;

/* --- Close the triangle --- */
points[3].x = 50;
points[3].y = 0;

/* --- Draw a polygon (don't fill it in) --- */
gdk_draw_polygon (drawable, gc, FALSE, points, 4);

/* --- Or... fill in the triangle --- */
gdk_draw_polygon (drawable, gc, TRUE, points, 4);
```

10.1.5　画多条线

能够使用两个函数来加快绘制多条线的速度。函数 gdk _ draw _ lines 函数取点的列表作为参数，并连接这些点。函数 gdk _ draw _ segments 取线段的数组作为参数，并按单独的线段绘制它们。下面的例子分别用这两个函数画前面的三角形。

```
/*
 * --- First try using gdk_draw_lines to draw the triangle
 */

/* --- Top point of triangle --- */
points[0].x = 50;
points[0].y = 0;

/* --- Bottom-left side of triangle --- */
points[1].x = 0;
points[1].y = 50;

/* --- Bottom-right side of triangle --- */
points[2].x = 100;
points[2].y = 50;

/* --- Close the triangle --- */
points[3].x = 50;
points[3].y = 0;

/* --- Draw lines between four points --- */
gdk_draw_lines (drawable, gc, points, 4);

/*
 * --- Then use gdk_draw_segments to draw the triangle
 */

/* --- Set up the segments of the triangle --- */
segment[0].x1 = 50;
segment[0].y1 = 0;
segment[1].x2 = 0;
segment[1].y2 = 50;

segment[1].x1 = 0;
segment[1].y1 = 50;
segment[1].x2 = 100;
```

```
segment[1].y2 = 50;

segment[2].x2 = 100;
segment[2].y2 = 50;
segment[2].x2 = 50;
segment[2].y2 = 0;

/* --- Draw three line segments --- */
gdk_draw_segments (drawable, gc, segment, 3);
```

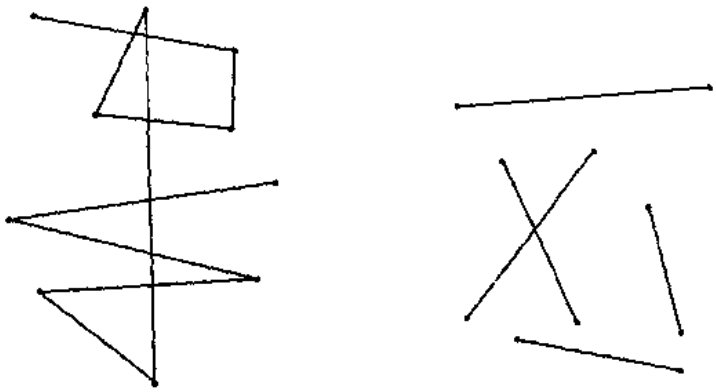


图 10.1 画多条线和线段

还可以用 gdk_draw_points 函数在可绘制构件内画多个点。用这种方法要比多次调用函数 gdk_draw_point 快。但是，从应用程序绘制单个像元都比较慢。如有可能，可以考虑使用像元图。

10.1.6 画圆和圆弧

函数 gdk_draw_arc 在可绘制区画圆或圆弧。需要对这个函数进行更多的解释，因为它有比较多的参数。函数的原型如下

```
gdk_draw_arc (GdkDrawable *drawable,     /* --- Drawing area --- */
              GdkGC       *gc,           /* --- How (color/font) --- */
              gint         filled,       /* --- Fill it in? --- */
              gint         x,            /* --- Left side --- */
              gint         y,            /* --- Top side --- */
              gint         width,
              gint         height,
              gint         angle1,       /* --- Start angle --- */
              gint         angle2);      /* --- End angle --- */
```

圆弧的开始角和终止角容易造成混淆。和其他圆弧函数不同，它不使用度或弧度表示。

角度必须转化为度乘以 64。时针指向 3 点表示 0 度。逆时针方向表示角度增加。如果你想从框的顶部到框的底部画一个圆弧，角度将从(90 * 64)到(180 * 64)。图 10.2 说明了该情况。

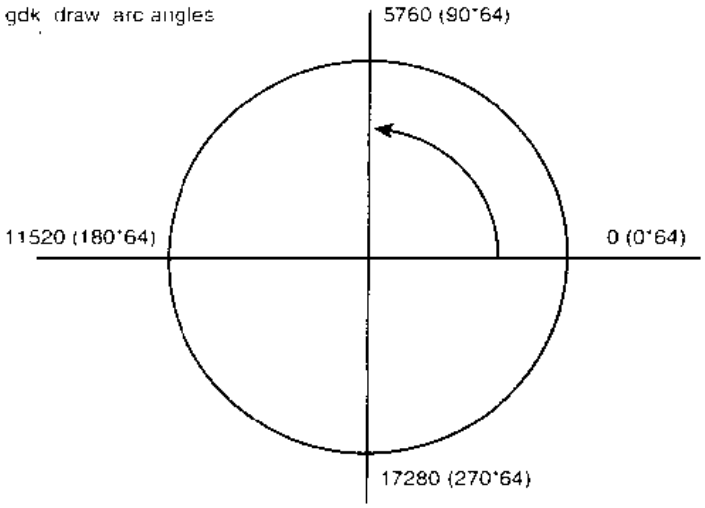


图 10.2　gdk _ draw _ arc

```
/* --- Draw a complete circle --- */
gdk_draw_arc (drawable, gc, FALSE, 0, 0, 30, 30, 0, (360 * 64));

/* --- Draw a 90 degree arc from 12 o'clock to 9 o'clock --- */
gdk_draw_arc (drawable, gc, FALSE, 0, 0, 30, 30, (90 * 64), (180 * 64));
```

10.2　显示正文

可以在可绘制区用 gdk _ draw _ string 函数绘制正文。函数除了要求使用可绘制区和 GdkGC 参数外，还要求使用字体参数。它在其他方面和其余的函数相同。

```
/* --- Display "Hey!" --- */
gdk_draw_string (drawable, font, gc, FALSE, 50, 50, "Hey!");
```

函数 gdk _ draw _ text 除了取字符串长度作为参数外，它完成函数 gdk _ draw _ string 同样的功能。

```
/* --- Display "Hey!" --- */
gdk_draw_text (drawable, font, gc, FALSE, 50, 50, szMessage, strlen (szMessage));
```

10.3　绘制像元图

当不能用画点和线解决问题时，函数 gdk _ draw _ pixmap 允许你将整个影像复制到可绘

制区。用这种方法复制影像到可绘制区是方便的。参阅本章"消除闪烁"部分以了解这一函数的更多信息。

```
/* --- Copy a 40x40 pixmap to the drawable widget --- */
gdk_draw_pixmap (drawable, gc, pixmap, 0, 0, 0, 0, 40, 40);
```

10.4 绘图区构件

绘图区构件是非常简单的构件。我们讨论过的大多数构件或者是容器,或者具有某些特征。绘图区构件只具有很少其他构件所具有的特征。这一构件具有尺寸的特征,可以用 gtk_drawing_area_size 函数设置这一特征。也就是这些了。这一构件的全部用途是提供可以使用 GDK 函数的区域,使得可以在构件内进行绘图。使用 gtk_drawing_area_new 函数建立可绘制构件。

```
/* --- Create the widget --- */
drawable = gtk_drawing_area_new ( );

/* --- Size the widget appropriately --- */
gtk_drawing_area_size (drawable, 200, 200);
```

10.5 可绘制区事件

对可绘制区重要的两个事件是 configure_event 和 expose_event。当建立可绘制区时或者重新设置可绘制区的尺寸时发出 configure_event 事件信号。当绘制区需要对它本身进行重新绘制时发出 expose_event。当绘制构件时,或者当其他窗口从构件上移开时(暴露构件),或者当应用程序产生重新绘制信息时也将发出 expose_event 事件信号。

10.6 简单的时钟应用

简单的时钟应用用来说明绘图区构件和 GDK 的使用。需要将时钟画在屏幕上(用圆弧和一些线条表示时钟的走动)。需要将时、分、秒针画上,并且每隔一秒更新一次。时钟每隔一秒用定时器对它本身进行更新,并重新绘制整个区域。图 10.3 表示你准备建立的时钟。

很明显,你需要运行定时器,并每隔一秒用当前时间重新绘制时钟。简单的时钟应用使用 gtk_timeout_add 函数设置定时器,每隔一秒调用 Repaint 函数一次。Repaint 函数每秒清除绘制区,并重新绘制整个时钟。时钟应用的代码如下:

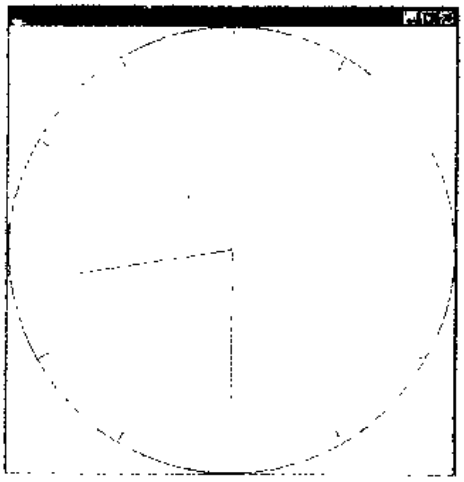


图 10.3　时钟应用

```
/*
 * Auth: Eric Harlow
 * Linux Application Development
 *
 *
 */

#include <gtk/gtk.h>
#include <time.h>

int radius;

static GdkColor clrBlue = { 0, 0x0000, 0x0000, 0xffff };

/*
 * DrawTickAt
 *
 * Draw a tick mark on the clock's face.  The tick marks
 * are drawn at the hour marks to help people read the
 * time on the clock.
 *
 * pixmap - drawing area
 * gc - pen
 * nHour - 1-12, what time to draw the tick mark
 * cx - width of clock
 * cy - height of clock
 */
```

```
voic DrawTickAt (GdkDrawable  * pixmap, GdkGC  * gc, int nHour, int cx, int cy)
{

    /* --- Convert time to radians --- */
    double dRadians = nHour * 3.14 / 6.0;

    /* --- Draw line from .95 * rad to 1.0 * rad --- */
    gdk_draw_line (pixmap, gc,
                   cx + (int) ((0.95 * radius * sin (dRadians))),
                   cy (int) ((0.95 * radius * cos (dRadians))),
                   cx + (int) ((1.0 * radius * sin (dRadians))),
                   cy-(int) ((1.0 * radius * cos (dRadians))));
}

/*
 * Repaint
 *
 * data - widget to repaint
 */
gint Repaint (gpointer data)
{
    GtkWidget *   drawing_area = (GtkWidget  * ) data;

int i;
    int midx, midy;
    int rHour;
    float dRadians;
    time_t now;
    struct tm * now_tm;
    GdkDrawable * drawable;

    /* --- Get drawing area window --- */
    drawable = drawing_area->window;

    /* --- Clear rectangle --- */

    gdk_draw_rectangle (drawable,
                        drawing_area->style->white_gc,
                        TRUE,
                        0, 0,
                        drawing_area->allocation.width,
                        drawing_area->allocation.height);
```

```
/* --- Get the midpoint --- */
midx = drawing_area->allocation.width / 2;
midy = drawing_area->allocation.height / 2;

/* --- Find smaller radius to make round clock --- */
radius = min (midx, midy);

/* --- Draw clock face (circle) --- */
gdk_draw_arc (drawable,
              drawing_area->style->black_gc,
              0,
              0, 0, midx + midx, midy + midy, 0, 360 * 64);

/* --- Draw tickmarks at each hour --- */
for (nHour = 1; nHour <= 12; nHour++) {

    DrawTickAt (drawable,
                drawing_area->style->black_gc,
                nHour, midx, midy);
}

/* --- Get time --- */
time (&now);

/* --- Convert time --- */
now_tm = localtime (&now);

/*
 * --- Draw the second hand
 */

/* --- Get radians from seconds --- */
dRadians = now_tm->tm_sec * 3.14 / 30.0;

/* --- Draw seconds --- */
gdk_draw_line (drawable, drawing_area->style->black_gc,
               midx, midy,
               midx + (0.9 * radius * sin (dRadians)),
               midy - (0.9 * radius * cos (dRadians)));

/*
```

```
     * --- Draw the minute
     */

    /* --- Get radians from minutes and seconds --- */
    dRadians = (now_tm->tm_min * 3.14 / 30.0) +
               (3.14 * now_tm->tm_sec / 1800.0);

    /* --- Draw minutes --- */
    gdk_draw_line (drawable, drawing_area->style->black_gc,
                   midx, midy,
                   midx + (int) (0.7 * radius * sin (dRadians)),
                   midy-(int) (0.7  * radius * cos (dRadians)));

    /*
     * --- Draw the hour hand
     */

    /* --- Calculate hours to the time in radians --- */
    dRadians = (now_tm->tm_hour % 12) * 3.14 / 6.0 +
               (3.14 * now_tm->tm_min / 360.0);

    /* --- Draw the hour hand --- */
    gdk_draw_line (drawable, drawing_area->style->black_gc,
                   midx, midy,
                   midx + (int) (radius * 0.5 * sin (dRadians)),
                   midy - (int) (radius * 0.5 * cos (dRadians)));

}

/*
 * quit
 *
 * exit gtk message loop
 */
void quit ()
{
    gtk_exit (0);
}


/*
 * main
```

```
 *
 * Program begins here.
 */
int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *drawing_area;
    GtkWidget *vbox;

    /* --- Initialize GTK --- */
    gtk_init (&argc, &argv);

    /* --- Create a top-level window --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- Get a packing box --- */
    vbox = gtk_hbox_new (FALSE, 0);

    /* --- Add packing box to window --- */
    gtk_container_add (GTK_CONTAINER (window), vbox);

    /* --- Make packing box visible --- */
    gtk_widget_show (vbox);

    /* --- Listen for the destroy --- */
    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                        GTK_SIGNAL_FUNC (quit), NULL);

    /* --- Create the drawing area --- */
    drawing_area = gtk_drawing_area_new ();

    /* --- Set the size --- */
    gtk_drawing_area_size (GTK_DRAWING_AREA (drawing_area), 200, 200);

    /* --- Add drawing area to packing box --- */
    gtk_box_pack_start (GTK_BOX (vbox), drawing_area, TRUE, TRUE, 0);

    /* --- Make drawing area visible --- */
    gtk_widget_show (drawing_area);

    /* --- Make the window visible --- */
    gtk_widget_show (window);
```

```
    /* - - Repaint every 1000ms (every second) --- */
    gtk_timeout_add (1000, Repaint, (gpointer) drawing_area);

    /*    - Start gtk message loop --- */
    gtk_main ();

    return 0;
}
```

还有什么问题?第一个问题是:在速度慢的机器上时钟将出现闪烁。即使在速度比较快的机器上,也会有一点闪烁。闪烁是由重新绘制子程序引起的。计算机每秒刷新屏幕许多次,如果机器紧接着程序清除绘图区后刷新屏幕,在屏幕上将短暂显示空白(或重新绘制)的绘图区。当然,当机器下一次刷新屏幕时,重新绘制工作应该已经完成,但已经太晚了。人的眼睛虽然看不到绘制的空白时钟,但能看到时钟的闪烁。幸运的是,当重新绘制时钟时,每秒只有一次闪烁的机会。绘制的任务愈复杂,机器在两次刷新屏幕之间完成重新绘制任务的可能性就愈小,用户看到闪烁的机会就愈大。闪烁不是好现象,需要设法消除它。

当重复移动窗口覆盖时钟,又从时钟上面将窗口移开。在这种情况下,将出现第二个问题。你最终将发现在移动窗口和在屏幕实际更新窗口内的影像之间有延迟。延迟不超过 1 秒,这是由于应用程序没有监听 expose_event 事件信号引起的。这一信号告诉应用程序:窗口暴露在外,应该对窗口进行重新绘制。好的 GDK 应用程序应该对 expose_event 增加回呼函数,并立即重新绘制时钟,不等到下 1 秒钟才开始重新绘制时钟。

10.6.1 消除闪烁

时钟应用的最大问题是重新绘制时出现的闪烁。消除闪烁的最好方法是使用双缓冲区技术。双缓冲区包括背景缓冲区。在屏幕上的影像并不直接在屏幕上进行绘制,所有的绘制工作都在背景缓冲区中进行。在本例中,可以建立和窗口同样大小的缓冲区。在缓冲区中而不是在窗口中绘制时钟。当在缓冲区中画好整个时钟后,再将缓冲区复制到窗口。由于传输缓冲区到窗口不包括清除屏幕,所以不出现闪烁。

要消除时钟闪烁的第一步是监听 configure_event 和 expose_event。当建立绘图区构件或改变绘图区构件的大小时产生 configure_event。这种情况提供建立与绘图区同样大小的背景缓冲区的机会。如果重新改变窗口大小,configure_event 的回呼函数按相应大小重新建立缓冲区。

10.6.1.1 configure_event

configure_event 的回呼函数建立背景缓冲区,它被用来绘制时钟而不影响显示在屏幕上的影像。背景缓冲区是一个像元图。可以用 gdk_pixmap_new 函数建立它,并得到和绘图区构件同样大小的像元图。如果绘图区构件的大小发生了变化,需要重新分配像元图。当改变

绘图区构件的大小时，调用 configure _ event 的回呼函数。这给你提供机会用新的尺寸建立新的像元图。由于不再需要旧的像元图，可以调用 gdk _ pixmap _ unref 函数告诉 GDK 不再引用旧的像元图。如果没有其他人引用它，它就被释放。最后得到的 configure _ event 函数如下：

```
static gint configure_event (GtkWidget *widget, GdkEventConfigure *event)
{
    /* --- Free background if we created it --- */
    if (pixmap) {
        gdk_pixmap_unref (pixmap);
    }

    /* --- Create a new pixmap with new size --- */
    pixmap = gdk_pixmap_new (widget->window,
                widget->allocation.width,
                widget->allocation.height,
                -1);

    return TRUE;
}
```

10.6.1.2　expose _ event

事实上，expose _ event 相当简单。由于在背景像元图中绘制时钟，expose _ event 函数的功能非常简单。它只需要用 gdk _ draw _ pixmap 函数将背景像元图复制到窗口就可以了。利用 expose _ event 消除了时钟的第二个问题：当另一个窗口从时钟上移开时没有立即重新绘制时钟。现在，当另一个窗口从时钟上移开时，应用程序的回呼函数接收 expose _ event 信号，并将最新的时钟影像（像元图）复制到窗口。expose _ event 与事件参数一起被发送，其中 area 字段表示需要重新绘制的暴露的窗口区域。当你有被覆盖的窗口，并将一个窗口从另一个窗口上移开时，就会出现这种情况。最快的解决方法是只重新绘制曾经被覆盖的部分。在本例中，必须强制重新绘制整个时钟。最终的 expose _ event 函数如下：

```
gint expose_event (GtkWidget *widget, GdkEventExpose *event)
{

    /* --- Copy pixmap to the window --- */
    gdk_draw_pixmap (widget->window,
        widget->style->fg_gc[GTK_WIDGET_STATE (widget)],
        pixmap,
        event->area.x, event->area.y,
        event->area.x, event->area.y,
        event->area.width, event->area.height);
```

```
    return FALSE;
}
```

你仍然还有不明白的地方。如果在背景像元图中进行绘制,那又如何更新屏幕?在时钟被画入像元图后,你只需要使用 expose _ event 回呼函数将像元图复制到窗口就行了。定义被暴露的区域(整个绘图区),并调用带有构件和需要更新的区域参数的 gtk _ widget _ draw 函数:

```
/* --- Define the area to be redrawn --- */
update_rect.x = 0;
update_rect.y = 0;
update_rect.width = drawing_area->allocation.width;
update_rect.height = drawing_area->allocation.height;

/* --- And then draw it (calls expose_event) --- */
gtk_widget_draw(drawing_area, &update_rect);
```

下面是实现双缓冲区的新的时钟应用程序。大部分代码和原来的程序相同。改变的是小部分,但效果却是明显的。

```
/*
 * Auth: Eric Harlow
 * File: dblclock.c
 * Linux application development
 *
 * Double buffered clock (no flickering)
 */

#include <gtk/gtk.h>
#include <time.h>
#include <stdlib.h>

/* --- Backing pixmap for drawing area --- */
static GdkPixmap *pixmap = NULL;

int radius;

/*
 * DrawTickAt
 *
 * Draw a tick mark on the clock's face.
 * pixmap - drawing area
 * gc - pen
```

```
 * nHour - 1-12, what time to draw the tick mark
 * cx - width of clock
 * cy - height of clock
 */
void DrawTickAt (GdkDrawable *pixmap, GdkGC *gc, int nHour, int cx, int cy)
{
    /* --- Convert time to radians --- */
    double dRadians = nHour * 3.14 / 6.0;

    /* --- Draw line from .95 * rad to 1.0 * rad --- */
    gdk_draw_line (pixmap, gc,
                   cx + (int) ((0.95 * radius * sin (dRadians))),
                   cy-(int) ((0.95 * radius * cos (dRadians))),
                   cx + (int) ((1.0 * radius * sin (dRadians))),
                   cy-(int) ((1.0 * radius * cos (dRadians))));
}


/*
 * Repaint
 *
 * data - widget to repaint
 */
gint Repaint (gpointer data)
{
    GtkWidget *     drawing_area = (GtkWidget *) data;
    GdkRectangle  update_rect;
    char buffer[88];
    int i;
    int prevx = 0;
    int prevy = 0;
    int x = 0;
    int y = 0;
    int midx, midy;
    int nHour;
    float dRadians;
    time_t now;
    struct tm *now_tm;

    /* --- Clear pixmap (background image) --- */
    gdk_draw_rectangle (pixmap,
              drawing_area->style->white_gc,
```

```
            TRUE,
            0, 0,
            drawing_area->allocation.width,
            drawing_area->allocation.height);

/* --- Calculate midpoint of clock --- */
midx = drawing_area->allocation.width / 2;
midy = drawing_area->allocation.height / 2;

/* --- Calculate radius --- */
radius = MIN (midx, midy);

/* --- Draw circle --- */
gdk_draw_arc (pixmap,
              drawing_area->style->black_gc,
              0,
              0, 0, midx + midx, midy + midy, 0, 360 * 64);

/* --- Draw tickmarks --- */
for (nHour = 1; nHour <= 12; nHour++) {

    DrawTickAt (pixmap, drawing_area->style->black_gc,
                nHour, midx, midy);
}

/* --- Get time --- */
time (&now);
now_tm = localtime (&now);

/*
 * --- Draw the second hand
 */

/* --- Get radians from seconds --- */
dRadians = now_tm->tm_sec * 3.14 / 30.0;

/* --- Draw seconds --- */
gdk_draw_line (pixmap, drawing_area->style->black_gc,
               midx, midy,
               midx + (0.9 * radius * sin (dRadians)),
               midy - (0.9 * radius * cos (dRadians)));
```

```
/*
 * --- Draw the minute
 */


/* --- Get radians from minutes and seconds --- */
dRadians = (now_tm->tm_min * 3.14 / 30.0) +
           (3.14 * now_tm->tm_sec / 1800.0);

/* --- Draw minutes --- */
gdk_draw_line (pixmap, drawing_area->style->black_gc,
               midx, midy,
               midx + (int) (0.7 * radius * sin (dRadians)),
               midy-(int) (0.7 * radius * cos (dRadians)));

/*
 * --- Draw the hour hand
 */

/* --- Calculate hours to the time in radians --- */
dRadians = (now_tm->tm_hour % 12) * 3.14 / 6.0 +
           (3.14 * now_tm->tm_min / 360.0);

/* --- Draw the hour hand --- */
gdk_draw_line (pixmap, drawing_area->style->black_gc,
               midx, midy,
               midx + (int) (radius * 0.5 * sin (dRadians)),
               midy - (int) (radius * 0.5 * cos (dRadians)));

/* --- Set the rect up
 * --- The entire pixmap is going to be copied
 * --- onto the window so the rect is configured
 * --- as the size of the window.
 */
update_rect.x = 0;
update_rect.y = 0;
update_rect.width = drawing_area->allocation.width;
update_rect.height = drawing_area->allocation.height;

/* --- And then draw it (calls expose_event) --- */
gtk_widget_draw (drawing_area, &update_rect);
}
```

```
/*
 * configure_event
 *
 * Create a new backing pixmap of the appropriate size
 * Of course, this is called whenever the window is
 * resized.  We have to free up things we allocated.
 */
static gint configure_event (GtkWidget *widget, GdkEventConfigure *event)
{
    /* --- Free background if we created it --- */
    if (pixmap) {
        gdk_pixmap_unref (pixmap);
    }

    /* --- Create a new pixmap with new size --- */
    pixmap = gdk_pixmap_new (widget->window,
                widget->allocation.width,
                widget->allocation.height,
                -1);

    return TRUE;
}


/*
 * expose_event
 *
 * When the window is exposed to the viewer or
 * the gdk_widget_draw routine is called, this
 * routine is called.  Copies the background pixmap
 * to the window.
 */
gint expose_event (GtkWidget *widget, GdkEventExpose *event)
{

    /* --- Copy pixmap to the window --- */
    gdk_draw_pixmap (widget->window,
          widget->style->fg_gc[GTK_WIDGET_STATE (widget)],
          pixmap,
          event->area.x, event->area.y,
          event->area.x, event->area.y,
```

```
            event->area.width, event->area.height);

    return FALSE;
}

/*
 * quit
 *
 * Get out of the application
 */
void quit ()
{
    gtk_exit (0);
}

/*
 * main
 *
 * Program begins here.
 */
int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *drawing_area;
    GtkWidget *vbox;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    vbox = gtk_hbox_new (FALSE, 0);
    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show (vbox);

    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                GTK_SIGNAL_FUNC (quit), NULL);

    /* --- Create the drawing area --- */
    drawing_area = gtk_drawing_area_new ();
    gtk_drawing_area_size (GTK_DRAWING_AREA (drawing_area), 200, 200);
    gtk_box_pack_start (GTK_BOX (vbox), drawing_area, TRUE, TRUE, 0);
```

```
    gtk_widget_show(drawing_area);

    /* --- Signals used to handle backing pixmap --- */
    gtk_signal_connect (GTK_OBJECT(drawing_area), "expose_event",
                        (GtkSignalFunc) expose_event, NULL);
    gtk_signal_connect (GTK_OBJECT(drawing_area),"configure_event",
                        (GtkSignalFunc) configure_event, NULL);

    /* --- Show the window --- */
    gtk_widget_show(window);

    /* --- Repaint every second --- */
    gtk_timeout_add(1000, Repaint, drawing_area);

    /* --- Call gtk_main loop --- */
    gtk_main();

    return 0;
}
```

10.7 系统监视

我们将重复使用原先用过的代码建立监视机器的小程序，它显示 CPU 和以太网连接状态的图形(见图 10.4)。唯一的问题是如何取得有关系统的信息。

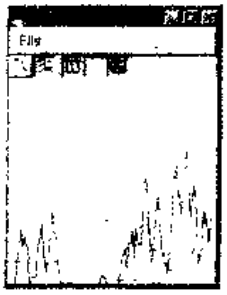


图 10.4 系统监视

10.7.1 使用 /proc 文件系统

有关 Linux 系统的许多现成可用的信息可以从 /proc 文件系统中得到。在 /proc 目录中的“文件”可以像普通文件一样被打开和进行分析，并能从中得到大量有关机器的信息。例如，/proc/loadavg 文件包含有关 CPU 负载的信息。如果你想知道在 /proc 文件系统中还有什

么其他文件，可以敲入 man proc 命令来得到这一系统中的所有文件的列表。

如果你用 cat 命令列出 /proc/loadavg 文件的内容，将得到类似如下的结果：

```
[brain@sharp /proc]$ cat loadavg
0.00 0.00 0.00 2/47 649
```

前面 3 个数字表示处理器的平均负载，分别表示最后 1 分钟，最后 5 分钟和最后 15 分钟的平均负载。如果你在 Linux 命令提示符的后面敲入 w 命令，将得到类似下面的结果：

```
4:15pm  up   3:44,  4 users,  load average: 0.00, 0.00, 0.00
```

在上面的行中，最后 3 个数字表示平均负载。看起来机器相当空闲。

另一个应用 /proc 系统的例子是 /proc/net/dev 文件，它包含每个网络设备发送和接收数据包数量的信息。在我的机器上，具有类似如下的结果：

```
Inter-|   Receive                                                |  Transmit
 face |bytes    packets errs drop fifo frame compressed multicast|bytes    packets errs drop fifo colls carrier compressed
    lo:    103       0    0    0    0     0          0         0      103       0    0    0    0     0       0          0
  eth0:   9600       0    0    0    0     0          0         0     6487       0    0    0    0     1       0          0
```

当然，发送和接收数据包的数量每次都有变化。如果过一段时间再敲入同一条命令，可能得到类似下面的结果：

```
Inter-|   Receive                                                |  Transmit
 face |bytes    packets errs drop fifo frame compressed multicast|bytes    packets errs drop fifo colls carrier compressed
    lo:    104       0    0    0    0     0          0         0      104       0    0    0    0     0       0          0
  eth0:   9639       0    0    0    0     0          0         0     6513       0    0    0    0     1       0          0
```

这里我需要监视的设备是 eth0:设备。eth0:设备是我的机器上的以太网卡。我想要知道在任何给定时间内有多少通信量通过这一网卡。通过与上次读文件时读入的数据包通信量总数比较，能算出从上次读文件起到现在为止总共有多少通信量。

10.7.2　说明

应用程序使用定时器每隔若干秒对屏幕更改一次。它读 /proc 文件系统得到有关系统的统计信息，并将这些信息用图形的方式显示出来。每种信息用不同的颜色表示。使用工具条按钮可以对图形进行配置。

在试验中发现网络使用图起伏很大，难以用它估计实际的通信量。使用最后的 N 个样本计算平均值，得到的图形起伏就比较小。用于系统监视的程序分成若干模块，每一模块完成其中一部分任务。对程序文件的说明从 frontend.c 开始。

10.7.2.1　frontend.c

frontend.c 建立应用程序的窗口，带像元图的工具条和小的菜单。它还有一些供其他函数使用的子程序。这些函数用来查询有关用户接口的信息。例如，可以调用 CPUBut-

ton15Down 函数来查询用户是否按下了工具条上的每隔 15 分钟显示 CPU 状态的按钮。这种方法比允许在此文件外使用 tool _ cpu15 变量更好(C 语言的数据封装)。注意，工具条按钮使用了不同的颜色，并用相同的颜色表示对应的信息的图形。可视颜色的提示有助于建立图形和按钮之间的联系，

```
/*
 * Sample GUI application front end.
 *
 * Auth: Eric Harlow
 * File: frontend.c
 *
 */

#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <gtk/gtk.h>

/*
 * --- Function prototypes
 */
GtkWidget *CreateDrawingArea();
static void CreateMainWindow();
void CreateToolbar(GtkWidget *vbox_main);
void SetToolbarButton(char *szButton, int nState);
void SelectMenu(GtkWidget *widget, gpointer data);
void DeSelectMenu(GtkWidget *widget, gpointer data);
void SetMenuButton(char *szButton, int nState);
GtkWidget *CreateWidgetFromXpm(GtkWidget *window, gchar **xpm_data);
GtkWidget *CreateMenuItem(GtkWidget *menu,
                          char *szName,
                          char *szAccel,
                          char *szTip,
                          GtkSignalFunc func,
                          gpointer data);
GtkWidget *CreateMenuCheck(GtkWidget *menu,
                           char *szName,
                           GtkSignalFunc func,
                           gpointer data);
GtkWidget *CreateSubMenu(GtkWidget *menubar, char *szName);
GtkWidget *CreateBarSubMenu(GtkWidget *menu, char *szName);
void Repaint();
```

```
/*
 * --- Global variables
 */
GtkWidget           *win_main;
GtkTooltips         *tooltips;
GtkAccelGroup       *accel_group;
GtkWidget           *toolbar;

static GtkWidget            *tool_cpu1;
static GtkWidget            *tool_cpu5;
static GtkWidget            *tool_cpu15;
static GtkWidget            *tool_network;
static GtkWidget            *tool_network_avg;


/*
 * --- Bitmap for "network"
 */
static const gchar *xpm_network[] = {
"16 16 4 1",
"  c None",
"B c #888888",
"                ",
"                ",
"  BBB           ",
"  BBBBBBB       ",
"  BBB   B       ",
"        B       ",
"        B       ",
"        B       ",
"        B       ",
"        B       ",
"        B       ",
"        B       ",
"        B  BBB  ",
"        BBBBBB  ",
"           BBB  ",
"                ",
"                ",
};
```

```
/*
 * --- Bitmap for "network"
 */
static const gchar * xpm_network_avg[] = {
"16 16 4 1",
"  c None",
"B c #000000",
"                ",
"                ",
"  BBB           ",
"  BBBBBBB       ",
"  BBB   B       ",
"        B       ",
"        B       ",
"        B       ",
"        B       ",
"        B       ",
"        B       ",
"        B  BBB  ",
"        BBBBBB  ",
"           BBB  ",
"                ",
"                ",
};


/*
 * --- Bitmap for "cpu" over last minute
 */
static const char * xpm_cpu1[] = {
"16 16 2 1",
"  c None",
"B c #FF0000",
"                ",
"   B B B B B B  ",
"  BBBBBBBBBBBBB ",
" BB           B ",
"  B     B     BB",
" BB    BB     B ",
"  B     B     BB",
" BB     B     B ",
```

```
"  B      B      BB",
" BB      B      B ",
"  B      B      BB",
" BB     BBB     B ",
"  B              BB",
"  BBBBBBBBBBBBBB ",
"   B B B B B B B  ",
"                  ",
};


/*
 * --- Bitmap for "cpu" over 5 minutes
 */
static const char * xpm_cpu5[] = {
"16 16 2 1",
"  c None",
"B c #00FF00",
"                  ",
"   B B B B B B  ",
"  BBBBBBBBBBBBB ",
" BB            B ",
"  B   BBBBB   BB",
" BB   B        B ",
"  B   B        BB",
" BB   BBBB     B ",
"  B       B    BB",
" BB       B    B ",
"  B   B   B    BB",
" BB    BBB     B ",
"  B              BB",
"  BBBBBBBBBBBBB ",
"   B B B B B B B  ",
"                  ",
};


/*
 * --- Bitmap for "cpu" over 15 minutes
 */
static const char * xpm_cpu15[] = {
"16 16 2 1",
```

```
"  c None",
"B c  # 0000FF",
"                ",
"   B B B B B B  ",
"  BBBBBBBBBBBBB ",
" BB           B ",
"  B  B  BBBBB BB",
" BB BB  B     B ",
"  B  B  B     BB",
" BB  B  BBBB  B ",
"  B  B      B BB",
" BB  B      B B ",
"  B  B  B   B BB",
" BB BBB  BBB  B ",
"  B           BB",
"  BBBBBBBBBBBBB ",
"   B B B B B B  ",
"                ",
};


/*
 * RepaintGraph
 *
 * Called when the user toggles buttons on the
 * toolbar to change the information being displayed
 * in the graph.  Called "Repaint" to force a repaint
 * of the graph.
 */
void RepaintGraph (GtkWidget *widget, gpointer data)
{
    Repaint ();
}

/*
 * NetworkButtonDown
 *
 * Is the network button down on the toolbar.
 */
int NetworkButtonDown ()
{
    return (GTK_TOGGLE_BUTTON (tool_network)->active);
```

```
}

/*
 * NetworkAvgButtonDown
 *
 * Is the network button down on the toolbar.
 */
int NetworkAvgButtonDown()
{
    return (GTK_TOGGLE_BUTTON(tool_network_avg)->active);
}

/*
 * CPUButton1Down
 *
 * Is the CPU-1 button down on the toolbar?
 */
int CPUButton1Down()
{
    return (GTK_TOGGLE_BUTTON(tool_cpu1)->active);
}

/*
 * CPUButton5Down
 *
 * Is the CPU-5 button down on the toolbar?
 */
int CPUButton5Down()
{
    return (GTK_TOGGLE_BUTTON(tool_cpu5)->active);
}

/*
 * CPUButton15Down
 *
 * Is the CPU-15 button down on the toolbar?
 */
int CPUButton15Down()

    return (GTK_TOGGLE_BUTTON(tool_cpu15)->active);
}
```

```
/*
 * EndProgram
 *
 * Exit from the program
 */
void EndProgram()
{
    gtk_main_quit();
}

/*
 * CreateMainWindow
 *
 * Create the main window and the menu/toolbar associated with it
 */
static void CreateMainWindow()
{
    GtkWidget *widget;
    GtkWidget *vbox_main;
    GtkWidget *menubar;
    GtkWidget *menu;
    GtkWidget *menuitem;

    /* --- Create the top window and size it. --- */
    win_main = gtk_window_new(GTK_WINDOW_TOPLEVEL);

    /* --- Don't let the window be resized. --- */
    gtk_window_set_policy(GTK_WINDOW(win_main), FALSE, FALSE, TRUE);

    /* --- Title --- */
    gtk_window_set_title(GTK_WINDOW(win_main), "system monitor");
    gtk_container_border_width(GTK_CONTAINER(win_main), 0);

    /* --- Create accel table --- */
    accel_group = gtk_accel_group_new();
    gtk_accel_group_attach(accel_group, GTK_OBJECT(win_main));

    /* --- Top-level window should listen for the destroy --- */
    gtk_signal_connect(GTK_OBJECT(win_main), "destroy",
            GTK_SIGNAL_FUNC(EndProgram), NULL);

    /* --- Create v-box for menu, toolbar --- */
```

```
    vbox_main = gtk_vbox_new(FALSE, 0);

    /*  -- Put up v-box --- */
    gtk_container_add(GTK_CONTAINER(win_main), vbox_main);

    gtk_widget_show(vbox_main);
    gtk_widget_show(win_main);

    /* --- Menu bar --- */
    menubar = gtk_menu_bar_new();
    gtk_box_pack_start(GTK_BOX(vbox_main), menubar, FALSE, TRUE, 0);
    gtk_widget_show(menubar);

    /* ------------------
       --- File menu ---
       ------------------ */
    menu = CreateBarSubMenu(menubar, "file");

    menuitem = CreateMenuItem(menu, "quit", "",
                    "that's more descriptive than quit?",
                    GTK_SIGNAL_FUNC(EndProgram), "quit");

    /* --- Create the toolbar --- */
    CreateToolbar(vbox_main);

    /* --- Create area to display information --- */
    widget = CreateDrawingArea();
    gtk_box_pack_start(GTK_BOX(vbox_main), widget, TRUE, TRUE, 0);
}


/*
 * CreateToolbar
 *
 * Create the toolbar buttons that will show or
 * hide items from being graphed in the drawing area.
 */
void CreateToolbar(GtkWidget *vbox_main)
{
    /* --- Create the toolbar and add it to the window --- */
    toolbar = gtk_toolbar_new(GTK_ORIENTATION_HORIZONTAL, GTK_TOOLBAR_ICONS);
    gtk_box_pack_start(GTK_BOX(vbox_main), toolbar, FALSE, TRUE, 0);
```

```
gtk_widget_show (toolbar);

/* --- Show network packet information --- */
tool_network = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar),
                        GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                        NULL,
                        "Network", "Network","Network",
                        CreateWidgetFromXpm (vbox_main, (gchar ** ) xpm_network),
                        (GtkSignalFunc) RepaintGraph,
                        NULL);

/* --- Show network packet information --- */
tool_network_avg = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar),
                        GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                        NULL,
                        "Network (avg)","Network (avg)","Network (avg)",
                        CreateWidgetFromXpm (vbox_main, (gchar ** ) xpm_network_avg),
                        (GtkSignalFunc) RepaintGraph,
                        NULL);

/*    Little gap --- */
gtk_toolbar_append_space (GTK_TOOLBAR (toolbar));

/* --- Show cpu utilization in last minute --- */
tool_cpu1 = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar),
                        GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                        NULL,
                        "CPU 1","CPU 1","CPU 1",
                        CreateWidgetFromXpm (vbox_main, (gchar ** ) xpm_cpu1),
                        (GtkSignalFunc) RepaintGraph,
                        NULL);

/* --- Show cpu utilization in last five minutes --- */
tool_cpu5 = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar),
                        GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                        NULL,
                        "CPU 5","CPU 5","CPU 5",
                        CreateWidgetFromXpm (vbox_main, (gchar ** ) xpm_cpu5),
                        (GtkSignalFunc) RepaintGraph,
                        NULL);

/* --- Show cpu utilization in last 15 minutes --- */
```

```
    tool_cpu15 = gtk_toolbar_append_element (GTK_TOOLBAR(toolbar),
                            GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                            NULL,
                            "CPU 15","CPU 15","CPU 15",
                            CreateWidgetFromXpm(vbox_main, (gchar **)gm_cpu15),
                            (GtkSignalFunc) RepaintGraph,
                            NULL);
}


/*
 * main
 *
 * --- Program bogins here
 */
int main(int argc, char *argv[])
{
    gtk_init (&argc, &argv);

    tooltips = gtk_tooltips_new();

    CreateMainWindow();

    gtk_main();

    return 0;
}
```

10.7.2.2　device.c

device.c 文件包含跟踪设备和用于表示图形的数据的一些子程序。将设备的信息保存在 GSList 中(参阅第 2 章,"GLIB")。这些子程序的编写特点是:可以用设备名称和一组值来保存不同的设备的信息,而不是每种设备使用不同的数据格式。由 device.c 使用的 typeDevice 在 device.h 中定义。

```
/*
 * device.h
 * Defines typDevice
 */
#define DELTA 1
#define ACTUAL 2
```

```
#define MAX_VALUES 200

typedef long typHistoryValue;

typedef struct {
    char              *sName;       /* --- Device name --- */
    int               nType;        /* --- Type - DELTA or ACTUAL --- */
    typHistoryValue nLast;          /* --- Last value (for DELTA) --- */
    typHistoryValue nMax;           /* --- Max value --- */
    typHistoryValue nValues[MAX_VALUES]; /* --- Values saved --- */
} typDevice;



/*
 * File: device.c
 * Auth: Eric Harlow
 *
 * Linux development
 */

#include <stdio.h>
#include <strings.h>
#include <gtk/gtk.h>
#include "device.h"

static GSList *devlist = NULL;

/*
 * LookupDevice
 *
 * Look up the device in the list of devices
 */
typDevice *LookupDevice (char *sName)
{
    GSList      *node;
    typDevice *dev;

    /* --- Go through the list of nodes. --- */
    for (node = devlist; node; node = node->next) {

        /* --- Get the data --- */
```

```
        dev = (typDevice *) node->data;

        /* --- This what we're looking for? --- */
        if (!strcmp (dev->sName, sName)) {
            return (dev);
        }
    }
    return (NULL);
}


/*
 * AddDevice
 *
 * Add a new device to the list of devices and
 * return it so it can be initialized.
 */
typDevice *AddDevice ()
{
    typDevice *dev;

    /* --- Create the new device --- */
    dev = (typDevice *) g_malloc (sizeof (typDevice));

    /* --- Add the item to the list --- */
    devlist = g_slist_append (devlist, dev);

    /* --- Return the new item --- */
    return (dev);
}


/*
 * UpdateDevice
 *
 * Updates the information about a device.  If the
 * device does not currently exist, it's created and
 * initialized.
 *
 * sDevName - device name."eth0:","cpu1", etc.
 * nValue - value at this time.
 * nType - DELTA/ACTUAL.
```

```
 *          ACTUAL means the value is "as is"
 *          DELTA means that the value is relative to
 *                the previous value and needs to be
 *                computed knowing that.
 * nMax - This is the normal max value to be displayed.
 */
void UpdateDevice (char * sDevName, long nValue, int nType, long nMax)
{
    typDevice   * dev;
    int         i;

    /* --- No name?  Get outta here! --- */
    if (sDevName == NULL) return;

    /* --- Look up the device by its name --- */
    dev = LookupDevice (sDevName);

    /* --- Do we have information about the device? --- */
    if (dev) {

        /* --- Just add a value. --- */
        NewValue (dev, nValue, FALSE);
    } else {
        /* --- Don't have the device! --- */

        /* --- Create a new device. --- */
        dev = AddDevice ();

        /* --- Initialize the values. --- */
        dev->sName = strdup (sDevName);
        dev->nType = nType;
        dev->nMax = nMax;

        /* --- Clear out values. --- */
        for (i = 0; i < MAX_VALUES; i++) {

            dev->nValues[i] = (typHistoryValue) 0;
        }

        /* --- Update with current position --- */
        NewValue (dev, nValue, TRUE);
    }
```

```
}


/*
 * UpdateExistingDevice
 *
 * If we know that the device exists, we use this
 * routine because it requires fewer parameters for me
 * to type and for the computer to push/pull from the
 * stack.
 */
void UpdateExistingDevice (char *sDevName, long nValue)
{
    typDevice  *dev;

    /* --- Name has to have a value. --- */
    if (sDevName == NULL) return;

    /* --- Look up the device name --- */
    dev = LookupDevice (sDevName);

    /* --- If the device exists... --- */
    if (dev) {

        /* --- Add the value. --- */
        NewValue (dev, nValue, FALSE);
    }
}


/*
 * NewValue
 *
 * Routine to add a value to the list of values
 * being stored for the device.  Also must shift the
 * values over by one every time we add another value.
 *
 * dev - device to add a value to
 * value - value to add to the device.
 * bInit - Is this the first value to be added to the device?
 */
void NewValue (typDevice *dev,
```

```
                typHistoryValue value,
                int bInit)
{
    int     i;

    /* --- Shift values down --- */
    for (i = MAX_VALUES-2; i >= 0; i--) {

        dev->nValues[i+1] = dev->nValues[i];
    }

    /* --- If this is NOT initialization --- */
    if (!bInit) {

        /* --- Add new value to the end --- */
        if (dev->nType == DELTA) {

            /* --- New value is a delta - diff between
             *      current value and previous value.
             */
            dev->nValues[0] = value - dev->nLast;
        } else {
            /* --- Use value as-is. --- */
            dev->nValues[0] = value;
        }
    }

    /* --- Store last value. --- */
    dev->nLast = value;

    /* --- Calculate the max value. --- */
    if (dev->nValues[0] > dev->nMax) {
        dev->nMax = dev->nValues[0];
    }
}
```

10.7.2.3 sys.c

sys.c 文件包含分析 /proc 文件(网络利用和 CPU 利用情况)的子程序。它调用在 device.c 中的子程序,更新每种设备的信息。用已有的代码作为例子,增加其他分析程序是方便的。增加新设备的过程包括:读入信息,分析它,并更新设备信息(在 device.c 中)。然后可以从通用数据结构中对它进行访问。

```
/*
 * File: sys.c
 * Auth: Eric Harlow
 *
 * Linux development
 */

#include <stdio.h>
#include <string.h>
#include <gtk/gtk.h>
#include "device.h"

#define SEPARATORS " :\n"
void GetPacketInfo();

/*
 * GetPacketInfo
 *
 * Get information about the packets going across the
 * ethernet.  Packet information is in /proc/net/dev if
 * the machine supports the /proc file system.
 */
void GetPacketInfo()
{

    char *szFilename = "/proc/net/dev";
    char szBuffer[132];
    int  nLineNo = 0;
    FILE *fp;
    char *sToken;
    char *sDevName;
    int  nWord;
    long nIncoming;
    long nOutgoing;
    typDevice *dev;

    /* --- Open the file to get the information --- */
    fp = fopen(szFilename, "r");
    if (fp) {

        /* --- While we have information to read --- */
        while (!feof(fp)) {
```

```
nLineNo++;

/* --- Read in a line of text --- */
fgets (szBuffer, sizeof (szBuffer), fp);

/* --- After 3rd line are the devices --- */
if (nLineNo >= 3) {

    nWord = 0;
    sDevName = NULL;
    sToken = strtok (szBuffer, SEPARATORS);
    while (sToken) {
        switch (nWord) {
            case 0:
                sDevName = sToken;
                break;
            case 1:
                nIncoming = atoi (sToken);
                break;
            case 6:
                nOutgoing = atoi (sToken);
                break;
        }

        nWord++;
        sToken = strtok (NULL, SEPARATORS);
    }

    /* --- Got a name --- */
    if (sDevName) {

        /* --- Look it up --- */
        dev = LookupDevice (sDevName);

        /* --- Add/Update it --- */
        if (dev) {
            UpdateExistingDevice (sDevName,
                (long) (nIncoming + nOutgoing));
        } else {
            UpdateDevice (sDevName,
                (long) (nIncoming + nOutgoing), DELTA, 1);
```

```
                    }
                }
                szBuffer[0] = (char) 0;
            }
        }

        /* --- All done --- */
        fclose (fp);
    } else {

        /* --- Uh... error! --- */
        printf ("Unable to open file %s.", szFilename);
    }
}



/*
 * GetCPUInfo
 *
 * Read in the /proc/loadavg file and parse for the cpu information.
 */
void GetCPUInfo ()
{
    static char * szFile = "/proc/loadavg";
    char szBuffer[88];
    FILE * fp;
    float cpu1, cpu2, cpu3;
    long lcpu1, lcpu2, lcpu3;
    typDevice * dev;

    /* --- Open the file with cpu info --- */
    fp = fopen (szFile, "r");

    /* --- If it's opened... --- */
    if (fp) {

        fgets (szBuffer, sizeof (szBuffer), fp);

        /* --- Get CPU utilization numbers --- */
        sscanf (szBuffer, "%f %f %f", &cpu1, &cpu2, &cpu3);
```

```
        /* ---Want range of 1-100+ instead of .01 to 1.0+ --- */
        lcpu1 = cpu1 * 100;
        lcpu2 = cpu2 * 100;
        lcpu3 = cpu3 * 100;

        /* ---Look up device--- */
        dev = LookupDevice ("cpu1");
        if (dev) {

            /* ---Already exists, update devices--- */

            UpdateExistingDevice ("cpu1", lcpu1);
            UpdateExistingDevice ("cpu5", lcpu2);
            UpdateExistingDevice ("cpu15", lcpu3);
        } else {

            /* ---First time, create devices--- */

            UpdateDevice ("cpu1", lcpu1, ACTUAL, 100);
            UpdateDevice ("cpu5", lcpu2, ACTUAL, 100);
            UpdateDevice ("cpu15", lcpu3, ACTUAL, 100);
        }
        /* ---Clean up.--- */
        fclose (fp);
    } else {
        printf ("Unable to open file %s.\n", szFile);
    }
}
```

10.7.2.4 graph.c

涉及绘图构件的部分都包含在本文件中。使用在 sys.c 中用过的虚构的设备名称查询信息。UpdateDevice 和 UpdateExistingDevice 建立一种设备,并将数字信息和它结合在一起。在 graph.c 中的子程序通过调用 device.c 的子程序访问这些信息。由于 graph.c 并不直接和/proc文件系统打交道,如果内核改变了(不太可能)或者你将应用移植到其他不支持/proc 的操作系统上,你就可以以后改变实施的细节(sys.c)。

许多子程序类似更改时钟的子程序。最大的差别在于这里在绘图中使用了颜色。在建立应用时产生了若干枝"笔"(实际是 GdkGC),允许使用特定的颜色进行绘图。预先分配这些笔有助于加速绘图工作(否则,你需要在每次重新绘制屏幕时建立新的 GdkGC)。在本例中,使用 GetPen 函数产生这些笔。和在 GetGdkGC 函数中产生这些笔的情况相比,它使代码更容易

阅读。

```
/*
 * File: graph.c
 * Auth: Eric Harlow
 *
 *
 */

#include <gtk/gtk.h>
#include "device.h"

void DrawDevice (GtkWidget *drawing_area, char *szName, GdkGC  *pen, int bAvg);
int NetworkButtonDown ();
int NetworkAvgButtonDown ();
int CPUButton15Down ();
int CPUButton5Down ();
int CPUButton1Down ();
void Repaint ();

GtkWidget *drawing_area;

typedef struct {

    GdkDrawable *pixmap;
    GdkGC *gc;

} typGraphics;

static typGraphics *g;
static GdkGC *penBlack = NULL;
static GdkGC *penRed = NULL;
static GdkGC *penBlue = NULL;
static GdkGC *penGreen = NULL;
static GdkGC *penGray = NULL;


/*
 * NewGraphics
 *
 * Create a new graphics data element to keep track
 * of the pixmap and the gc.
```

```
 */
typGraphics * NewGraphics ()
{
    typGraphics * gfx;

    /* --- Allocate the space --- */
    gfx = (typGraphics * ) g_malloc (sizeof (typGraphics));

    /* --- Initialize --- */
    gfx->gc = NULL;
    gfx->pixmap = NULL;

    /* --- Hand back, ready to use. --- */
    return (gfx);
}


/*
 * GetPen
 *
 * Get a pen using the GdkColor passed in.  The "pen"
 * (just a GdkGC) is created and returned ready for
 * use.
 */
GdkGC *GetPen (GdkColor * c)
{
    GdkGC * gc;

    /* --- Create a gc --- */
    gc = gdk_gc_new (g->pixmap);

    /* --- Set the foreground to the color --- */
    gdk_gc_set_foreground (gc, c);

    /* --- Return it. --- */
    return (gc);
}


/*
 * NewColor
 *
```

```
 * Create and allocate a GdkColor with the color
 * specified in the parameter list.
 */
GdkColor *NewColor (long red, long green, long blue)
{
    /* --- Get the color --- */
    GdkColor *c = (GdkColor *) g_malloc (sizeof (GdkColor));

    /* --- Fill it in. --- */
    c->red = red;
    c->green = green;
    c->blue = blue;

    gdk_color_alloc (gdk_colormap_get_system (), c);

    return (c);
}


/*
 * UpdateAndRepaint
 *
 * Routine to poll for the latest stats on network
 * traffic and cpu utilization and repaint the
 * screen with the information
 */
gint UpdateAndRepaint (gpointer data)
{
    /* --- Get information about networking --- */
    GetPacketInfo ();

    /* --- Get information about the CPU --- */
    GetCPUInfo ();

    /* --- Repaint. --- */
    Repaint ();

    return (1);
}


/*
```

```
 * Repaint
 *
 * Update the screen with the latest numbers.
 */
void Repaint ()
{
    GdkRectangle      update_rect;

    /* --- Clear pixmap so we can draw on it. --- */
    gdk_draw_rectangle (g->pixmap,
                drawing_area->style->white_gc,
                TRUE,
                0, 0,
                drawing_area->allocation.width,
                drawing_area->allocation.height);

    /* --- If they want to see the actual network usage --- */
    if (NetworkButtonDown ()) {
        DrawDevice (drawing_area, "eth0:", penGray, 0);
    }

    /* --- If they want to see the averaged network usage --- */
    if (NetworkAvgButtonDown ()) {
        DrawDevice (drawing_area, "eth0:", penBlack, 1);
    }
    /* --- If they want to see the 15 minute cpu util --- */
    if (CPUButton15Down ()) {
        DrawDevice (drawing_area, "cpu15", penBlue, 0);
    }
    /* --- If they want to see the 5 minute cpu util --- */
    if (CPUButton5Down ()) {
        DrawDevice (drawing_area, "cpu5", penGreen, 0);
    }
    /* --- If they want to see the 1 minute cpu util --- */
    if (CPUButton1Down ()) {
        DrawDevice (drawing_area, "cpu1", penRed, 0);
    }

    /* --- Update the screen with the background pixmap --- */
    update_rect.x = 0;
    update_rect.y = 0;
    update_rect.width = drawing_area->allocation.width;
```

```
    update_rect.height = drawing_area->allocation.height;

    gtk_widget_draw (drawing_area, &update_rect);
}


/*
 * DrawDevice
 *
 * Draw a graph with the device's information
 *
 * drawing_area - widget
 * szName - name of the device being monitored
 * pen - GC with the color information
 * bAvg - Average plot indicator.  True => do some averaging
 */
void DrawDevice (GtkWidget *drawing_area, char *szName, GdkGC *pen, int bAvg)
{
    typDevice  *dev;
    int    prevx = 0;
    int    prevy = 0;
    int    x = 0;
    int    y = 0;
    int    i;
    int    nLast;

    /* --- Look up the device by the name --- */
    dev = LookupDevice (szName);

    /* --- If we found it --- */
    if (dev) {

        /* --- If they're averaging --- */
        if (bAvg) {
            nLast = MAX_VALUES-4;
        } else {
            nLast = MAX_VALUES;
        }

        /* --- Go across the widget and plot --- */
        for (i = 0; i < drawing_area->allocation.width && i < nLast; i++) {
```

```
            x = i;
            if (dev->nMax != 0) {

                if (bAvg) {

                    y = ((dev->nValues[i] +
                          dev->nValues[i+1] +
                          dev->nValues[i+2] +
                          dev->nValues[i+3] +
                          dev->nValues[i+4]) *
                         drawing_area->allocation.height) /
                         (dev->nMax * 5);
                } else {
                    y = (dev->nValues[i] *
                     drawing_area->allocation.height) / dev->nMax;
                }

                y = drawing_area->allocation.height - y;
            } else {
                y = 1;
            }

            if (i == 0) {
                prevx = x;
                prevy = y;
            }

            /* --- Draw a line from previous point to current pt. --- */
            gdk_draw_line (g->pixmap,
                           pen,
                           prevx, prevy,
                           x, y);

            /* --- Next previous point is this point. --- */
            prevx = x;
            prevy = y;
        }
    } else {

        /* --- Should never occur. --- */
        printf ("it's NULL (%s)\n", szName);
    }
}
```

```
/*
 * configure_event
 *
 * Called when the drawing area is created and every
 * time it's resized.
 * Create a new backing pixmap of the appropriate size
 */
static gint configure_event (GtkWidget  *widget, GdkEventConfigure *event)
{
    if (g == NULL) {
        g = NewGraphics ();
    }

    /* --- Free the pixmap --- */
    if (g->pixmap) {
        gdk_pixmap_unref (g->pixmap);
    }

    /* --- Create a new pixmap --- */
    g->pixmap = gdk_pixmap_new (widget->window,
                                widget->allocation.width,
                                widget->allocation.height,
                                -1);

    /* --- If we don't have our pens yet --- */
    if (penBlack == NULL) {

        /* --- Get our colorful pens. --- */
        penBlack = GetPen (NewColor (0, 0, 0));
        penRed = GetPen (NewColor (0xffff, 0, 0));
        penBlue = GetPen (NewColor (0, 0, 0xffff));
        penGreen = GetPen (NewColor (0, 0xffff, 0));
        penGray = GetPen (NewColor (0x9000, 0x9000, 0x9000));
    }

    /* --- Clear it out --- */
    gdk_draw_rectangle (g->pixmap,
                        widget->style->white_gc,
                        TRUE,
                        0, 0,
                        widget->allocation.width,
```

```
                    widget->allocation.height);

    return TRUE;
}

/*
 * expose_event
 *
 * Redraw the screen using the pixmap
 */
static gint expose_event (GtkWidget *widget, GdkEventExpose *event)
{
  /* --- Copy it right over. --- */
  gdk_draw_pixmap (widget->window,
          widget->style->fg_gc[GTK_WIDGET_STATE (widget)],
          g->pixmap,
          event->area.x, event->area.y,
          event->area.x, event->area.y,
          event->area.width, event->area.height);

  return FALSE;
}


/*
 * quit
 *
 * Quit the application
 */
void quit ()
{
    gtk_exit (0);
}


/*
 * CreateDrawingArea
 *
 * Create the drawing area widget and return it.
 */
GtkWidget *CreateDrawingArea ()
{
```

```
    /* --- Create the drawing area  --- */
    drawing_area = gtk_drawing_area_new();

    /* --- Give it a nice size --- */
    gtk_drawing_area_size(GTK_DRAWING_AREA(drawing_area), 200, 200);

    /* --- Make it visible --- */
    gtk_widget_show(drawing_area);

    /* --- We need the expose_event and the configure_event --- */
    gtk_signal_connect(GTK_OBJECT(drawing_area), "expose_event",
                       (GtkSignalFunc) expose_event, NULL);
    gtk_signal_connect(GTK_OBJECT(drawing_area),"configure_event",
                       (GtkSignalFunc) configure_event, NULL);

    /* --- Call the function every 2 seconds. --- */
    gtk_timeout_add(2000, UpdateAndRepaint, drawing_area);

    /* --- Return widget so it can be placed on the screen. --- */
    return(drawing_area);
}
```

10.8　小结

将 GDK 和绘图区构件结合在一起使你能够建立包含图形的复杂应用。GDK 有许多用在绘图区构件中的绘制简单图形的子程序。在绘图区中使用双缓冲区技术有助于在应用中消除闪烁，但增加了一些工作量。显示图形信息的应用应该经常使用双缓冲区技术来消除闪烁。

第 11 章　式样、颜色、字体、光标和引用

式样(style)影响如何使用颜色和字体绘制构件。每种式样包含若干种颜色和单一的字体。式样是 GTK + 的一部分,而颜色和字体则是 GDK(绘图工具集)的一部分。虽然颜色是 GDK 的一部分,GTK + 具有可以用在应用中的颜色选择对话框,它可供用户选择颜色。与此类似,GTK + 还有字体选择对话框,可供需要它的应用使用。这些对话框构件类似文件选择对话框,并为需要让用户选择颜色和字体的开发人员提供通用的用户接口。GDK 还提供将鼠标改变为内置光标或定制光标的能力。引用(reference)帮助 GTK + 保持对构件使用情况的跟踪。GTK + 认为构件不再需要使用时释放构件。但是,在某些情况下你需要阻止这一操作。

11.1　式样(颜色和字体)

在 GTK + 中的每个构件都有与它结合在一起的式样,用式样来确定如何在屏幕上绘制构件。基本式样包含颜色组合和一种字体。GTK + 的应用通常共用同一种式样。这就是为什么所有构件使用相同的颜色组合和字体的原因。构件可以用它自己单独的式样,但是给每个构件单独一种式样并不是好的做法。在一页上用多种式样显示不同的构件可能会使应用显得混乱。特别是如果构件使用不同的颜色,情况就更为严重。最好将式样的数量减到最少(或者只用一种),而把时间和精力用在提高应用的功能上。

虽然在一种式样中只定义一种字体,但式样可以有多种颜色。构件可以处于几种不同的状态:NORMAL、ACTIVE、PRELIGHT、SELECTED 和 INSENSITIVE。每种状态用单独的颜色组合表示。

NORMAL	绘制对象的常用方式。
ACTIVE	对象处于活动状态。对按钮讲,当按钮按下时为活动状态。
PRELIGHT	通常表示鼠标移到构件上,颜色提示用户点击构件。
SELECTED	表示选择了对象
INSENSITIVE	使构件成为不敏感(也就是不可选)。相当 Microsoft 的术语"禁止"。

对每种状态定义一组颜色,用来表示构件所处的状态。你可以在按钮构件上看到这一特征。按钮构件通常处于 NORMAL 状态。当鼠标移到按钮构件上时,构件进入 PRELIGHT 状态,这将导致按钮显示 PRELIGHT 颜色。如果按下了按钮,则用按钮构件的 ACTIVE 颜色显示按钮。不是所有构件都支持每一种状态。

11.1.1　颜色

颜色被定义为 GdkColor。颜色由红、绿、蓝(RGB)值组成。这些值确定每种成分对颜色的贡献。

```
/* --- Red defined as - lot of red, no green, no blue --- */
GdkColor color = {0, 0xffff, 0x0000, 0x0000};
```

在建立 GdkColor 结构后，应该用 gdk_color_alloc 函数从系统分配颜色。gdk_color_alloc 试图用硬件能够支持颜色满足对颜色的要求。函数要求使用 colormap 和 color 作为参数。如果你只用少量的颜色，可以利用系统的 colormap。它可以从 gdk_colormap_get_system 函数的返回值得到。在分配了颜色以后，就可以在这些颜色的基础上用更有趣的颜色建立新的式样。

11.1.2　使用式样

可以使用 gtk_widget_get_default_style 函数提取用于应用的默认式样。直接修改这一式样是不好的做法，你可以取得式样的拷贝再对它进行修改。函数 gtk_style_copy 建立式样的拷贝，你可以对它进行修改。定义式样颜色的 GdkStyle 结构包含前景颜色数组(fg)，背景颜色数组(bg)和的正文颜色数组(text)。要使按钮在黑底上显示白字，可以按如下方式进行修改：

```
/* --- Define the colors --- */
GdkColor white = {0, 0xffff, 0xffff, 0xffff};
GdkColor black = {0, 0x0000, 0x0000, 0x0000};

/* --- Allocate the colors --- */
gdk_color_alloc (gdk_colormap_get_system (), &white);
gdk_color_alloc (gdk_colormap_get_system (), &black);

/* --- Get default style --- */
defstyle = gtk_widget_get_default_style ();

/* --- Make a copy for ourselves. --- */
style = gtk_style_copy (defstyle);

/* --- Modify the color --- */
style->fg[NORMAL] = white;
style->text[NORMAL] = white;
style->bg[NORMAL] = black;
```

在建立式样后，可以用它来修改任何已有构件的式样。函数 gtk_widget_set_style 设置已有构件的式样。函数 gtk_widget_set_style 设置作为参数传递给它的构件的式样。但是，如果构件是一个容器，函数并不设置它的子构件的式样。可以用函数 gtk_container_foreach 来改变构件的式样以及构件内所有子构件的式样：

```
void SetStyleRecursively (GtkWidget *widget, gpointer data)
```

```
{
    GtkStyle * style;

    /* --- Get the style --- */
    style = (GtkStyle * ) data;

    /* --- Set the style of the widget --- */
    gtk_widget_set_style (widget, style);

    /* --- If it might have children widgets --- */
    if (GTK_IS_CONTAINER (widget)) {

        /* --- Set all the children's styles too. --- */
        gtk_container_foreach (GTK_CONTAINER (widget),
                               SetStyleRecursively, style);
    }
}
```

代替在建立构件后再改变构件的式样,你可以方便地改变与构件建立在一起的式样。函数 gtk_widget_push_style 对所有建立的构件设置当前的式样。在 gtk_widget_push_style 函数将式样"压入"堆栈后建立任何构件。当用完后,应该调用 gtk_widget_pop_style 函数将式样从堆栈中移去。由于在堆栈中实现式样,任何时候在堆栈中都可以有多个式样存在,但是只使用堆栈顶部的式样。如果使用 gtk_widget_pop_style 函数从堆栈顶部弹出式样,在堆栈中的下一个式样就成为建立构件时使用的式样。在改变构件的式样后,将向构件发送"style_set"信号。

下面的例子说明如何用式样改变按钮的颜色或改变构件的默认的式样。程序显示 6 个按钮。前 4 个按钮放在用特定颜色组合的对话框内(提示:Christmas 的颜色组合是绿和红)。第 5 个按钮导致所有按钮改变颜色。最后的按钮改变所有的式样,它开始以不同的颜色出现。事实上,当鼠标移过按钮时显示 PRELIGHT 是黄色而不是默认的颜色。点击 Button Go Wild 按钮将更换初始的式样为其他式样。这里没有包括 CallButton 函数,它包括在其他文件中。它弹出对话框和显示一条消息(见图 11.1)。你可以看一下它的源代码,这是重要的部分。

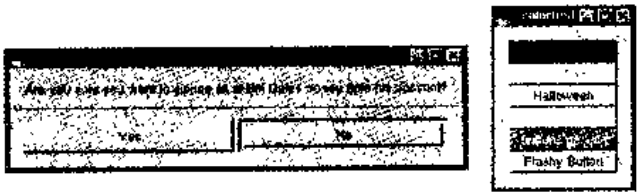


图 11.1 改变颜色对话框和它的结果

```
/*
 * File: buttor.c
 * Auth: Eric Harlow
 *
 * Show how to modify styles - specifically colors
 *
 * ---NORMAL, PRELIGHT, ACTIVE, INSENSITIVE, SELECTED
 */

#include <gtk/gtk.h>
#include <stdlib.h>

GtkWidget *button;
GtkStyle *styleRed;

/*
 * ---Define colors
 */
GdkColor red = {0, 0xffff, 0x0000, 0x0000};
GdkColor blue = {0, 0x0000, 0x0000, 0xffff};
GdkColor green = {0, 0x0000, 0xffff, 0x0000};
GdkColor yellow = {0, 0xffff, 0xffff, 0x0000};
GdkColor purple = {0, 0xffff, 0x0000, 0xffff};
GdkColor orange = {0, 0xffff, 0x9999, 0x0000};
GdkColor cyan = {0, 0x0000, 0xffff, 0xffff};
GdkColor black = {0, 0x0000, 0x0000, 0x0000};
GdkColor white = {0, 0xffff, 0xffff, 0xffff};

/* ---Create a list of colors to use for the
 *      random color button.
 */
GdkColor colorlist[] = {
    {0, 0xffff, 0x0000, 0x0000},
    {0, 0x0000, 0x0000, 0xffff},
    {0, 0x0000, 0xffff, 0x0000},
    {0, 0xffff, 0xffff, 0x0000},
    {0, 0xffff, 0x0000, 0xffff},
    {0, 0xffff, 0x9999, 0x0000},
    {0, 0x0000, 0xffff, 0xffff},
    {0, 0x0000, 0x0000, 0x0000},
    {0, 0xffff, 0xffff, 0xffff}
};
```

```
/* --- Calculate number of buttons. --- */
static int numColors = sizeof (colorlist) / sizeof (GdkColor);

/* --- Create styles for the dialog windows --- */
GtkStyle * styleChristmas;
GtkStyle * styleHalloween;
GtkStyle * styleStPatricks;
GtkStyle * styleUSIndependence;

/* --- vbox for the buttons --- */
GtkWidget * vbox;

/*
 * CreateColorfulStyle
 *
 * Create a style using the colors passed in.
 * Set the foreground color, the text color, and the
 * background color.  Note thet this makes all the
 * states be the same color.
 *
 * fg - foreground color
 * text - text color
 * bg - background color
 */
GtkStyle * CreateColorfulStyle (GdkColor fg,
                                GdkColor text,
                                GdkColor bg)
{
    GtkStyle * defstyle;
    GtkStyle * style;
    int i;

    /* --- Get the default style --- */
    defstyle = gtk_widget_get_default_style ();

    /* --- Make a copy of it. --- */
    style = gtk_style_copy (defstyle);

    /* --- Set the colors for each state --- */
    for (i = 0; i < 5; i++) {
```

```
        /* --- Set the colors for the style --- */
        style->fg[i] = fg;
        style->text[i] = text;
        style->bg[i] = bg;
    }

    /* --- All done, here's new style --- */
    return (style);
}


/*
 * RandomStyle
 *
 * Create a random style using the list of colors
 * above.  There is no checking to see if the colors
 * would be a bad choice, like having background color the
 * same as the foreground color.
 */
GtkStyle *RandomStyle ()
{
    GtkStyle *style;

    /* --- Create a randomly colored style --- */
    style = CreateColorfulStyle (
              colorlist[random () % numColors],
              colorlist[random () % numColors],
              colorlist[random () % numColors]);

    return (style);
}


/*
 * CreateFunkyStyles
 *
 * Create styles for the buttons.  Each style has a different set of
 * colors that will be assigned to buttons to make each have a different
 * color.
 */
void CreateFunkyStyles ()
{
    /* --- Red on green --- */
```

```
    styleChristmas = CreateColorfulStyle (red, red, green);

    /* --- Orange on black --- */
    styleHalloween = CreateColorfulStyle (orange, orange, black);

    /* --- Green on white --- */
    styleStPatricks = CreateColorfulStyle (green, green, white);

    /* --- red and white on blue --- */
    styleUSIndependence = CreateColorfulStyle (red, white, blue);
}


/*
 * CloseApp
 *
 * Shut down GTK.
 */
void CloseApp (GtkWidget *widget, gpointer gdata)
{
    gtk_main_quit ();
}


/*
 * button_was_clicked
 *
 * event handler called when the button is clicked.
 */
void ButtonClicked (GtkWidget *widget, gpointer gdata)
{
    GtkStyle *style;

    /* --- Get the style from the callback --- */
    style = (GtkStyle *) gdata;

    /* --- Push colorful new style - now it's the default --- */
    gtk_widget_push_style (style);

    /* --- Show dialog using fancy style --- */
    CallButton (widget, (gpointer) 2);
```

```
    /* --- Remove the style. --- */
    gtk_widget_pop_style ();
}



/*
 * StyleSet
 *
 * Prints a message when the widget's style is modified.
 */
void StyleSet (GtkWidget *widget, gpointer data)
{

    printf ("Style set\n");
}



/*
 * SetStyleRecursively
 *
 * Set the widget's style to the style (data) and make sure
 * that all the children (if it's a container) are also set
 * to that particular style.
 */
void SetStyleRecursively (GtkWidget *widget, gpointer data)
{
    GtkStyle *style;

    /* --- Get the style --- */
    style = (GtkStyle *) data;

    /* --- Set the style of the widget --- */
    gtk_widget_set_style (widget, style);

    /* --- If it may have children widgets --- */
    if (GTK_IS_CONTAINER (widget)) {

        /* --- Set all the children's styles too. --- */
        gtk_container_foreach (GTK_CONTAINER (widget),
                               SetStyleRecursively, style);
    }
```

```
}


/*
 * WildButton
 *
 * Create a wildly colored button using the color table we
 * defined.
 */
void WildButton (GtkWidget *widget, gpointer data)
{
    GtkStyle *style;

    /* --- Pick a random style --- */
    style = RandomStyle ();

    /* --- Set the style of the widget --- */
    SetStyleRecursively (widget, (gpointer) style);
}


/*
 * ColorizeButtons
 *
 * Set the color of the children widgets to a wild (and random)
 * style.
 */
ColorizeButtons (GtkWidget *widget, gpointer data)
{
    /* --- All children get their style reset --- */
    gtk_container_foreach (GTK_CONTAINER (vbox), WildButton, NULL);
}


/*
 * CreateColorDialogButton
 *
 * Create a button in the vbox.  The style is going to be
 * associated with the button in the callback so that the
 * dialog created when the button is pressed is in the
```

```
 * style.
 */
CreateColorDialogButton (GtkWidget *vbox, char *label, GtkStyle *style)
{
    GtkWidget *button;

    /* --- Create a button with a label --- */
    button = gtk_button_new_with_label (label);

    /* --- Set up the style in the callback of the "clicked" --- */
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                GTK_SIGNAL_FUNC (ButtonClicked), (gpointer) style);

    /* --- Set up the style in the callback of the "style_sheet" --- */
    gtk_signal_connect (GTK_OBJECT (button), "style_sheet",
                GTK_SIGNAL_FUNC (StyleSet), (gpointer) style);

    /* --- The vbox contains the button. --- */
    gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);

    /* --- Make the button visible --- */
    gtk_widget_show (button);
}



/*
 * CreateColorizeButton
 *
 * Create a button that changes all the button colors.
 */
void CreateColorizeButton (GtkWidget *vbox, char *label)
{
    GtkWidget *button;

    /* --- Create a button --- */
    button = gtk_button_new_with_label (label);

    /* --- Set up the signal handler --- */
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                GTK_SIGNAL_FUNC (ColorizeButtons), NULL);
```

```
    /* --- The main window contains the button. --- */
    gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);

    /* --- Make the button visible --- */
    gtk_widget_show (button);
}


/*
 * CreateFlashyButton
 *
 */
void CreateFlashyButton (GtkWidget * vbox, char * label)
{
    GtkStyle * defstyle;
    GtkStyle * style;

    defstyle = gtk_widget_get_default_style ();
    style = gtk_style_copy (defstyle);

    style->fg[GTK_STATE_NORMAL] = purple;
    style->text[GTK_STATE_NORMAL] = purple;
    style->bg[GTK_STATE_NORMAL] = cyan;

    style->fg[GTK_STATE_PRELIGHT] = green;
    style->text[GTK_STATE_PRELIGHT] = green;
    style->bg[GTK_STATE_PRELIGHT] = blue;

    style->fg[GTK_STATE_ACTIVE] = orange;
    style->text[GTK_STATE_ACTIVE] = orange;
    style->bg[GTK_STATE_ACTIVE] = yellow;

    gtk_widget_push_style (style);

    /* --- Create a new button --- */
    button = gtk_button_new_with_label (label);

    /* --- The main window contains the button. --- */
    gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);

    /* --- Make the button visible --- */
    gtk_widget_show (button);
```

```
    /* --- Remove the style so it's not the default style --- */
    gtk_widget_pop_style ();
}


/*
 * main
 *
 * Program begins here
 */
int main (int argc, char *argv[])
{
    GtkWidget *window;

    /* --- Initialize gtk, handle command-line parameters --- */
    gtk_init (&argc, &argv);

    /* --- Create a window in gtk - window is NOT visible yet. --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                GTK_SIGNAL_FUNC (CloseApp), NULL);

    /* --- Put some room around the objects in container --- */
    gtk_container_border_width (GTK_CONTAINER (window), 15);

    vbox = gtk_vbox_new (FALSE, 0);

    /* --- Allocate the styles --- */
    CreateFunkyStyles ();

    /* --- Create buttons --- */
    CreateColorDialogButton (vbox, "Christmas", styleChristmas);
    CreateColorDialogButton (vbox, "St. Patricks", styleStPatricks);
    CreateColorDialogButton (vbox, "Halloween", styleHalloween);
    CreateColorDialogButton (vbox, "US Independence", styleUSIndependence);

    CreateColorizeButton (vbox, "Buttons go wild");

    CreateFlashyButton (vbox, "Flashy Button");
```

```
    /* --- Now, make the window visible --- */
    gtk_widget_show (vbox);
    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show (window);

    /* --- This is the event loop in gtk.  Wait for "quit" --- */
    gtk_main ();

    /* --- Exit status code. --- */
    return 0;
}
```

11.1.3 选择颜色对话框

类似选择文件对话框为用户提供选择文件的标准界面,选择颜色对话框为用户提供选择颜色的标准方法。选择颜色对话框有若干帮助用户选择颜色的构件(见图 11.2)

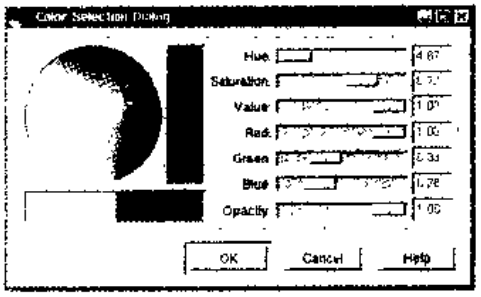


图 11.2 选择颜色构件

使用 gtk_color_selection_dialog_new 函数建立选择颜色对话框。对话框的 OK 和 Cancel 按钮可以有处理按钮事件的回呼函数与它们结合在一起。

```
/* --- New dialog box --- */
window = gtk_color_selection_dialog_new ("Color Selection Dialog");

/* --- Want to know when 'ok' button pressed --- */
gtk_signal_connect (
        GTK_OBJECT (GTK_COLOR_SELECTION_DIALOG (window)->ok_button),
        "clicked",
        GTK_SIGNAL_FUNC (OkButtonClicked),
        colorInfo);

/* --- Destroy dialog box when "dancel" is clicked --- */
gtk_signal_connect_object (
```

```
GTK_OBJECT (GTK_COLOR_SELECTION_DIALOG (window)->cancel_button),
  "clicked",
  GTK_SIGNAL_FUNC(gtk_widget_destroy),
  GTK_OBJECT (window));
```

"color_changed"信号表示何时在对话框中发生了颜色变化。gtk_color_selection_set_update_policy 函数影响何时发送变更消息的方式。这些方式有：GTK_UPDATE_CONTINUOUS、GTK_UPDATE_DISCONTINUOUS 和 GTK_UPDATE_DELAYED。

当在对话框中改变颜色时 GTK_UPDATE_CONTINUOUS 连续发送变更消息。当鼠标按钮释放时 GTK_UPDATE_DISCONTINUOUS 发送变更消息。当鼠标按钮释放时或当鼠标在颜色范围内停留一段时间后 GTK_UPDATE_DELAYED 发送变更消息。

```
/* --- Want to know when the color changes --- */
gtk_signal_connect (
    GTK_OBJECT (GTK_COLOR_SELECTION_DIALOG (window)->colorsel),
    color_changed,
    GTK_SIGNAL_FUNC (ColorSelectionChanged),
    Window);

/* --- Need to know immediately when the color changes --- */
/* --- flood with messages - we got a fast machine --- */
gtk_color_selection_set_update_policy (
  GTK_COLOR_SELECTION (GTK_COLOR_SELECTION_DIALOG (window)->colorsel),
  GTK_UPDATE_CONTINUOUS);
```

当信号处理程序需要知道在对话框中的当前颜色，处理程序需要经过几个步骤。首先，信号处理程序需要取得颜色对话框的 colorsel 成分。其次，需要提取颜色值数组（定义为 gdoubles）。这些值在 0 到 1 的范围内，所以需要将它们乘以 0xffff 转换到 0～0xffff 的范围。除了定义颜色外，返回值还定义颜色不透明度（opacity）。可以用函数 gtk_color_selection_set_opacity 配置不透明度，而且它将在颜色选择对话框中产生处理不透明度的附加构件。不透明度由数组的第 4 个值定义。

```
GtkColorSelection *colorsel;
gdouble dacolor[4];

/* --- Get the color window to ask for the color --- */
colorsel = GTK_COLOR_SELECTION (cs->colorsel);

/* --- Get the color values in the widget --- */
gtk_color_selection_get_color (colorsel, dacolor);
```

```
/* --- color is a GdkColor --- */
color->red = dacolor[0] * 0xffff;
color->green = dacolor[1] * 0xffff;
color->blue = dacolor[2] * 0xffff;

/* --- Opacity would be dacolor[3], if we set it in the dialog box ---*/

/* --- Color is now retrieved for use --- */
```

将这些放在一起,说明了如何使用选择颜色对话框改变构件的颜色。下面的程序允许用户点击按钮和选择新颜色来为按钮取得新颜色。当按下 OK 按钮时,改变按钮的式样以包含新的颜色。本例包括两个文件 colorbutton.c 和 colordiag.c。

11.1.3.1 colorbutton.c

colorbutton.c 文件建立主窗口、控制按钮和改变颜色。

```
/*
 * File: colorbutton.c
 *
 */
#include <gtk/gtk.h>
#include <stdlib.h>

GtkWidget *button;
GtkWidget *vbox;

/*
 * CreateBackgroundStyle
 *
 * Create a style using the colors passed in.
 * Set the foreground color, the text color, and the
 * background color.  Note that this makes all the
 * states the same color.
 *
 * fg - foreground color
 * text - text color
 * bg - background color
 */
GtkStyle *CreateBackgroundStyle(GdkColor bg)
{
    GtkStyle *defstyle;
    GtkStyle *style;
```

```
    int i;

    /* --- Get the default style --- */
    defstyle = gtk_widget_get_default_style();

    /* --- Make a copy of it. --- */
    style = gtk_style_copy(defstyle);

    /* --- Set the colors for each state --- */
    for(i = 0; i < 5; i++) {

        /* --- Set the colors for the style --- */
        style->bg[i] = bg;
    }

    /* --- All done, here's new style --- */
    return(style);
}


/*
 * NewStyle
 *
 * Create a random style using the list of colors
 * above.  There is no checking to see whether the colors
 * would be a bad choice.
 */

GtkStyle *NewStyle(GdkColor c)
{
    GtkStyle *style;

    /* --- Create a style from the color --- */
    style = CreateBackgroundStyle(c);

    /* --- Return the style --- */
    return(style);
}


/*
 * CloseAppWindow
```

```
 *
 * Close down the application
 */
gint CloseAppWindow (GtkWidget *widget, gpointer gdata)
{
    g_print ("Quitting...\n");
    gtk_main_quit ();
    return (FALSE);
}


/*
 * SetStyleRecursively
 *
 * Set the style on the current widget and on any
 * children widgets.
 */
void SetStyleRecursively (GtkWidget *widget, gpointer data)
{
    GtkStyle *style;

    /* --- Get the style --- */
    style = (GtkStyle *) data;

    /* --- Set style on current widget --- */
    gtk_widget_set_style (widget, style);

    /* --- If it has other widgets --- */
    if (GTK_IS_CONTAINER (widget)) {

        /* --- Set the style on those widgets too --- */
        gtk_container_foreach (GTK_CONTAINER (widget),
                               SetStyleRecursively, style);
    }
}


/*
 * button_was_clicked
 *
 * event handler called when the button is clicked.
 * We can do this because the color dialog box is modal here
```

```
 * and doesn't return until a color is retrieved.
 */
void ButtonClicked (GtkWidget *widget, gpointer gdata)
{
    GtkStyle *style;
    GdkColor color;

    /* --- Call the color dialog box to get a color --- */
    GetDialogColor (&color);

    /* --- Create a style based on the color --- */
    style = NewStyle (color);

    /* --- Set the style of the widget based on that new style */
    SetStyleRecursively (widget, (gpointer) style);
}

/*
 * CreateButton
 *
 * Create a button and add it to the vbox. Set up the
 * event handler to "ButtonClicked"
 */
CreateButton (GtkWidget *vbox, char *label)
{
    GtkWidget *button;

    /* --- Create a button --- */
    button = gtk_button_new_with_label (label);

    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                GTK_SIGNAL_FUNC (ButtonClicked), NULL);

    /* --- The main window contains the button.  --- */
    gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);

    /* --- Make the button visible --- */
    gtk_widget_show (button);
}


/*
```

```
 * main
 *
 * Program begins here.
 */
int main (int argc, char *argv[])
{
    GtkWidget *window;

    /* --- Initialize gtk, handle command-line parameters --- */
    gtk_init (&argc, &argv);

    /* --- Create a window --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- Need to know when window closes --- */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                GTK_SIGNAL_FUNC (CloseAppWindow), NULL);

    /* --- Put some room around objects in the container --- */
    gtk_container_border_width (GTK_CONTAINER (window), 15);

    /* --- Create vertical packing box --- */
    vbox = gtk_vbox_new (FALSE, 0);

    /* --- Create a button --- */
    CreateButton (vbox, "pick Color");

    /* --- Now, make the window visible --- */
    gtk_widget_show (vbox);
    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show (window);

    /* --- Do not return until "quit" --- */
    gtk_main ();

    /* --- Exit status code. --- */
    return 0;
}
```

11.1.3.2 colordiag.c

colordiag.c 文件显示颜色对话框,并给调用程序返回一种颜色。

```
/*
 * File: colordiag.c
 *
 * Show the color dialog box to allow someone to pick
 * a color for the button.
 */
#include <gtk/gtk.h>


/*
 * Structure to eliminate global variables
 */
typedef struct {

    GtkWidget *dialog;
    GdkColor *color;

} typColorDialogInfo;



/*
 * CloseDialog
 *
 * Close the modal dialog box using the gtk_main_quit because it
 * was made modal by the gtk_main function.  Also release
 * the grab and free the memory.
 */
void CloseDialog (GtkWidget *w, typColorDialogInfo *di)
{
    gtk_main_quit ();
    gtk_grab_remove (w);
    g_free (di);
}



/*
 * OkButtonClicked
 *
 * Get the color currently selected in the dialog box and
 * get the rgb components out.  Set the GdkColor in the
 * colorInfo structure to those values.
```

```
 */
void OkButtonClicked (GtkWidget *w,
                      typColorDialogInfo *colorInfo)
{
    GtkColorSelectionDialog *cs;
    GtkColorSelection *colorsel;
    gdouble color[4];

    /* --- Get the dialog box and the color --- */
    cs = (GtkColorSelectionDialog *) colorInfo->dialog;
    colorsel = GTK_COLOR_SELECTION (cs->colorsel);

    /* --- Get the color and assign it to the GdkColor --- */
    gtk_color_selection_get_color (colorsel, color);
    colorInfo->color->red = color[0] * 0xffff;
    colorInfo->color->green = color[1] * 0xffff;
    colorInfo->color->blue = color[2] * 0xffff;

    /* --- Destroy the dialog box --- */
    gtk_widget_destroy (GTK_WIDGET (cs));
}

/*
 * ColorSelectionChanged
 *
 * Notified when the color has changed within the dialog box.
 */
void ColorSelectionChanged (GtkWidget *w,
                            GtkColorSelectionDialog *cs)
{
    GtkColorSelection *colorsel;
    gdouble color[4];

    /* --- Get the color from the selection --- */
    colorsel = GTK_COLOR_SELECTION (cs->colorsel);
    gtk_color_selection_get_color (colorsel, color);

g_print ("The color has been changed! \n");
}


/*
```

```
 * GetDialogColor
 *
 * Display a modal dialog box and get the user to select a
 * color that can be sent back to the user.
 */
void GetDialogColor (GdkColor *color)
{
    static GtkWidget *window = NULL;
    typColorDialogInfo *colorInfo;

    /* --- New dialog box --- */
    window = gtk_color_selection_dialog_new ("color Selection Dialog");

    /* --- Allocate memory for the colorinfo and populate it --- */
    colorInfo = g_malloc (sizeof (typColorDialogInfo));
    colorInfo->dialog = window;
    colorInfo->color = color;

    gtk_color_selection_set_opacity (
        GTK_COLOR_SELECTION (GTK_COLOR_SELECTION_DIALOG (window)->colorsel),
        TRUE);

    gtk_color_selection_set_update_policy (
        GTK_COLOR_SELECTION (GTK_COLOR_SELECTION_DIALOG (window)->colorsel),
        GTK_UPDATE_CONTINUOUS);

    /* --- Want to know when window is destroyed --- */
    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                        GTK_SIGNAL_FUNC (CloseDialog),
                        colorInfo);

    /* --- Want to know when color changes --- */
    gtk_signal_connect (
        GTK_OBJECT (GTK_COLOR_SELECTION_DIALOG (window)->colorsel),
        "color_changed",
        GTK_SIGNAL_FUNC (ColorSelectionChanged),
        window);

    /* --- Want to know when 'ok' button pressed --- */
    gtk_signal_connect (
        GTK_OBJECT (GTK_COLOR_SELECTION_DIALOG (window)->ok_button),
        "clicked",
```

```
        GTK_SIGNAL_FUNC (OkButtonClicked),
        colorInfo);

    /* --- Destroy dialog box when "cancel" is clicked --- */
    gtk_signal_connect_object (
        GTK_OBJECT (GTK_COLOR_SELECTION_DIALOG (window)->cancel_button),
        "clicked",
        GTK_SIGNAL_FUNC(gtk_widget_destroy),
        GTK_OBJECT (window));

    /* --- Show window --- */
    gtk_widget_show (window);

    /* --- Grab --- */
    gtk_grab_add (window);

    /* --- This makes it modal in a loop until quit is called --- */
    gtk_main ();
}
```

11.1.4 字体

除了在所有状态中都使用同一种字体外,可以使用类似改变颜色的方式改变字体。可以用名称或者使用字体选择对话框装入字体(见图 11.3)。在不使用选择字体对话框的情况下使用新字体需要知道字体的名称和使用 gtk_font_load 函数。在装入字体后,可以设置式样的字体字段。

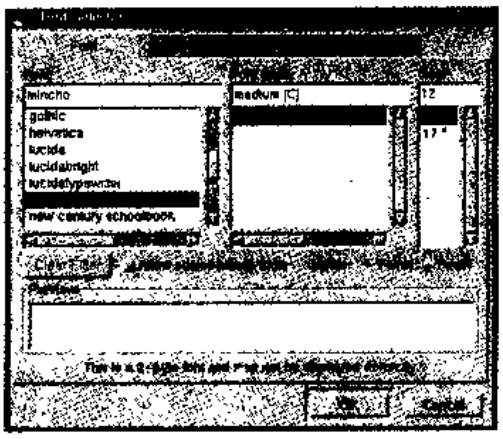


图 11.3 选择字体对话框

```
/* --- Load up the font --- */
    font = gdk_font_load (szFont);

    /* --- This style uses the new font --- */
    style->font = font;
```

现在的问题成为如何确定用于系统的有效字体名称。为了显示系统的所有可用字体，可以使用 Xlib 的 XlistFonts 函数。这一函数要求用模式(pattern)作为参数。用模式来确定从字体名称列表中返回哪一种字体。为了匹配所有的可用字体，使用 -* 作为模式。* 匹配任何东西(如文件系统)，因而匹配任何字体。字体具有类似如下的名称(这些是我的系统上的部分字体)。

```
-adobe-courier-medium-r-normal--8-80-75-75-m-50-iso8859-1
-adobe-helvetica-bold-o-normal--10-100-75-75-p-60-iso8859-1
-daewoo-gothic-medium-r-normal--16-120-100-100-c-160-ksc5601.1987-0
-daewoo-mincho-medium-r-normal--24-170-100-100-c-240-ksc5601.1987-0
-isas-fangsong ti-medium-r-normal--16-160-72-72-c-160-gb2312.1980-0
-isas-song ti-medium-r-normal--24-240-72-72-c-240-gb2312.1980-0
-jis-fixed-medium-r-normal--24-230-75-75-c-240-jisx0208.1983-0
-misc-fixed-bold-r-normal--13-120-75-75-c-80-iso8859-1
-adobe-times-medium-i-normal--12-120-75-75-p-63-iso8859-1
```

下面的短程序可以用来显示你的系统中的所有字体。

```
/*
 * File: DisplayFonts.c
 *
 * Show all fonts available on the system.
 *
 * Needs to be run from an X Window display.
 */

#include <gtk/gtk.h>
#include <gdk/gdkx.h>
#include <X11/Xlib.h>

/*
 * If they have more than this many fonts, they have
 * too many fonts anyway.
 */
#define MAX_FONTS 30000

/*
```

```
 * main.c
 *
 */
int main (int argc, char * argv[])
{
    int nFonts;
    char ** szaFontNames;
    int i;

    /* --- Initialize GTK+.  Needed for the GDK_DISPLAY call --- */
    gtk_init (&argc, &argv);

    /* --- Get the font names --- */
    szaFontNames = XListFonts (GDK_DISPLAY (), "*", MAX_FONTS, &nFonts);

    /* --- Check number retrieved --- */
    if (nFonts == MAX_FONTS) {

        /* --- They have a lot of fonts on their system. --- */
        printf ("Many fonts on your system.  Not displaying all.");
    }

    /* --- Display all the fonts --- */
    for (i = 0; i < nFonts; i++) {

        /* --- Get the name --- */
        printf ("%s\n", szaFontNames[i]);
    }

    /* --- Clean up. --- */
    XFreeFontNames (szaFontNames);
}
```

你可以修改选择颜色对话框的例子,使它从选择字体对话框为按钮取得一种字体。除了用户选择字体代替选择颜色外,这个例子和颜色选择对话框的例子几乎完全相同。对话框返回字体名称。用字体名称作为参数调用 gtk_font_load 函数装入字体和改变按钮的式样。图 11.3 显示使用中的选择字体对话框。注意:字体并不一定是英文字体。

```
/*
 * File: fontbutton.c
 *
 */
```

```
#include <gtk/gtk.h>
#include <stdlib.h>

GtkWidget *button;
GtkWidget *vbox;

gchar *GetFont();


/*
 * NewStyle
 *
 * Create a new style with the font passed in.
 */

GtkStyle *NewStyle(GdkFont *f)
{
    GtkStyle *style;
    GtkStyle *defstyle;
    int i;

    /* --- Get the default style --- */
    defstyle = gtk_widget_get_default_style();

    /* --- Make a copy of it. --- */
    style = gtk_style_copy(defstyle);

    style->font = f;

    /* --- return the style --- */
    return(style);
}


/*
 * CloseAppWindow
 *
 * Close down the application
 */
gint CloseAppWindow(GtkWidget *widget, gpointer gdata)
{
    g_print("Quitting...\n");
```

```
    gtk_main_quit ();
    return (FALSE);
}


/*
 * SetStyleRecursively
 *
 * Set the style on the current widget and on any
 * children widgets.
 */
void SetStyleRecursively (GtkWidget *widget, gpointer data)
{
    GtkStyle *style;

    /* --- Get the style -- */
    style = (GtkStyle *) data;

    /* --- Set style on current widget --- */
    gtk_widget_set_style (widget, style);

    /* --- If it has other widgets --- */
    if (GTK_IS_CONTAINER (widget)) {

        /* --- Set the style on those widgets too --- */
        gtk_container_foreach (GTK_CONTAINER (widget),
                               SetStyleRecursively, style);
    }
}


/*
 * button_was_clicked
 *
 * event handler called when the button is clicked.
 * We can do this because the font dialog box is modal here
 * and doesn't return until a font is retrieved.
 */

void ButtonClicked (GtkWidget *widget, gpointer gdata)
{
    GtkStyle *style;
```

```
    GdkColor color;
    char * szFont;
    GdkFont  * font;

    /* --- Call the font dialog box to get a font --- */

    szFont = GetFont ();

    printf ("GetFont = %s \n", szFont);
    font = gdk_font_load (szFont);
    g_free (szFont);

    /* --- Create a style based on the font --- */

    style = NewStyle (font);

    /* --- Set the style of the widget based on that new style
*/
    SetStyleRecursively (widget, (gpointer) style);
}

/*
 * CreateButton
 *
 * Create a button and add it to the vbox. Set up the
 * event handler to "ButtonClicked"
 */
CreateButton (GtkWidget * vbox, char * label)
{
    GtkWidget * button;

    /* --- Create a button --- */
    button = gtk_button_new_with_label (label);

    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                GTK_SIGNAL_FUNC (ButtonClicked), NULL);

    /* --- The main window contains the button.  --- */
    gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);

    /* --- Make the button visible --- */
    gtk_widget_show (button);
```

```
}


/*
 * main
 *
 * Program begins here.
 */
int main (int argc, char *argv[])
{
    GtkWidget *window;

    /* --- Initialize gtk, handle command-line parameters --- */
    gtk_init (&argc, &argv);

    /* --- Create a window --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- Need to know when window closes --- */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                GTK_SIGNAL_FUNC (CloseAppWindow), NULL);

    /* --- Put some room around objects in the container --- */
    gtk_container_border_width (GTK_CONTAINER (window), 15);

    /* --- Create vertical packing box --- */
    vbox = gtk_vbox_new (FALSE, 0);

    /* --- Create buttons --- */
    CreateButton (vbox, "Pick Font");
    CreateButton (vbox, "The fox jumped over the lazy dog");

    /* --- Now, make the window visible --- */
    gtk_widget_show (vbox);
    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show (window);

    /* --- Do not return until "quit" --- */
    gtk_main ();

    /* --- Exit status code. --- */
    return 0;
```

建立独占的字体对话框的程序如下。程序需要调用 GetFont 函数，并在 gchar * szFontName 中返回字体名称。当不使用时应该释放它。

```
/*
 * File: gfontsel.c
 *
 * Front end to the font dialog box.  Creates a modal window
 * to select a font.  Use the GetFont () function to pick a name.
 *
 */

#include <gtk/gtk.h>
#include "gtkfontsel.h"

gchar  * szFontName = NULL;

/*
 * OkClicked
 *
 * Ok button was clicked
 */
void OkClicked (GtkWidget *widget, GtkWidget * fontsel)
{

    /* --- Typecast to correct type --- */
    GtkFontSelectionDialog * fsd = GTK_FONT_SELECTION_DIALOG (fontsel);

    /* --- Get the font name --- */
    szFontName = gtk_font_selection_dialog_get_font_name (fsd);

    /* --- Display the font name --- */
    printf ("Ok clicked - %s\n", szFontName);

    /* --- Destroy the font selection window --- */
    gtk_widget_destroy (fontsel);
}


/*
 * Quit
 *
```

```
 * Quit from the program
 */
void Quit (GtkWidget * widget, gpointer data)
{
    gtk_main_quit ();
}


/*
 * GetFont
 *
 * Show the font selection dialog box to allow user to
 * pick a font.  Return a gchar * with the font
 * name.
 */
gchar * GetFont ()
{
    GtkWidget * widget;
    GtkFontSelectionDialog * fontsel;
    szFontName = NULL;

    /* --- Create the font selection dialog box --- */
    widget = gtk_font_selection_dialog_new ("Font Selector");

    /* --- Typecast to correct type --- */
    fontsel = GTK_FONT_SELECTION_DIALOG (widget);

    /* --- Callback for the clicked button --- */
    gtk_signal_connect (GTK_OBJECT (fontsel->ok_button), "clicked",
                        GTK_SIGNAL_FUNC (OkClicked), fontsel);

    /* --- Callback for the cancel button --- */
    gtk_signal_connect_object (GTK_OBJECT (fontsel->cancel_button), "clicked",
                               GTK_SIGNAL_FUNC (gtk_widget_destroy),
                               GTK_OBJECT (fontsel));

    /* --- Destroy signal --- */
    gtk_signal_connect (GTK_OBJECT (fontsel), "destroy",
                        GTK_SIGNAL_FUNC (Quit), fontsel);


    /* --- Show the dialog box --- */
```

```
gtk_widget_show(widget);

/* --- Modal - wait until destroyed --- */
gtk_main();

/* --- Return the font name --- */
return(szFontName);
```

11.2　光标

可以用光标给用户提供反馈信息。当光标移过输入字段时(I光束光标)或者当重新改变窗口尺寸时(箭头形光标),你可以看到光标形状的变化。这种反馈信息有时是很重要的。GTK+带有一组完整的内置光标。可以在 GTK+安装的 CDK 目录下看到内置光标的完整列表。在 gdkcursors.h 中定义你能够使用的内置光标。这样的例子有 CDK_ARROW 和 GDK_CLOCK。使用 gdk_cursor_new 函数建立内置光标,以内置光标的识别符作为参数。可以将光标指定给窗口,而不是给构件。然而,利用窗口的构件成分,建立窗口和构件的联系是方便的。

```
/* --- Create a new cursor --- */
cursor = gdk_cursor_new(GDK_CROSS);

/* --- Assign cursor to the button --- */
gdk_window_set_cursor(button->window, cursor);
```

除了在 GTK+中内置的光标数量有限外,它的使用足够方便。假定你需要建立定制光标。可以像建立按钮图标一样用 xpm 数据建立定制光标。这里使用的技巧是影像必须使用3种颜色:用于前景的颜色、用于背景的颜色和透明的颜色。必须将 xpm 数据转换成两个 GtkBitmaps:一个用于前景,一个用于背景。在转换后,就很容易建立光标。例如,使用下列 xpm 影像:

```
static char * cursor_hand[] = {
"32 32 3 1",
"       c None",
".      c #000000",
"+      c #FFFFFF",
"                                ",
"                                ",
"              ...               ",
"            .+++.   ...         ",
"            .+++. .+++.   ...   ",
```

```
"   ...      .+++.  .+++.  .+++.        ",
"  .+++.     .+++. .+++.  .+++.         ",
"   .+++.    .+++. .+++. .+++.          ",
"    .+++.   .+++. .+++. .+++.          ",
"     .+++. .+++..+++. .+++.            ",
"      .+++..+++..+++. .+++.            ",
"       .+++.+++..+++..+++.             ",
"   ...  .++++++++++++++++.             ",
"  .+++. .+++++++++++++++.              ",
"  .++++..+++++++++++++++.              ",
"   ..+++++++++++++++++++.              ",
"     .+++++-++++++++++++.              ",
"      .+++++++++++++++++.              ",
"       ..++++++++++++++.               ",
"         .++-+++++++++.                ",
"          .++++++++++.                 ",
"           ..........                  ",
"                                       ",
"                                       ",
"                                       ",
"                                       ",
"                                       ",
"                                       ",
"                                       ",
"                                       ",
"                                       "};
```

这里你有一个带 3 种颜色的、高 32 个像元和宽 32 个像元的手的影像。在影像中定义的颜色实际被忽略。利用这些数据,你可以用来自 gnumeric(你喜欢开放源代码的软件吗?)的函数提取前景和掩模(mask)影像。这一函数取 xpm 数据,并返回两个位图。一个是前景影像,另一个是掩模影像。

```
void
create_bitmap_and_mask_from_xpm (GdkBitmap ** bitmap,
                                 GdkBitmap ** mask,
                                 gchar ** xpm)
{
        int height, width, colors;
        char pixmap_buffer [(32 * 32)/8];
        char mask_buffer [(32 * 32)/8];
        int x, y, pix;
```

```
    int transparent_color, black_color;

    sscanf (xpm[0], "%d %d %d %d", &height, &width, &colors, &pix);

    g_assert (height == 32);
    g_assert (width  == 32);
    g_assert (colors == 3);

    transparent_color = ' ';
    black_color = '.';

    for (y = 0; y < 32; y++){
        for (x = 0; x < 32;){
            char value = 0, maskv = 0;

            for (pix = 0; pix < 8; pix++, x++){
                if (xpm[4+y][x] != transparent_color){
                    maskv |= 1 << pix;

                    if (xpm[4+y][x] != black_color){
                        value |= 1 << pix;
                    }
                }
            }
            pixmap_buffer[(y * 4 + x/8)-1] = value;
            mask_buffer[(y * 4 + x/8)-1] = maskv;
        }
    }
    *bitmap = gdk_bitmap_create_from_data (NULL, pixmap_buffer, 32, 32);
    *mask   = gdk_bitmap_create_from_data (NULL, mask_buffer, 32, 32);
}
```

现在可以用这一函数建立光标。当然，应该用 gdk_cursor_new_from_pixmap 函数而不是用 gdk_cursor_new 函数建立光标。这一函数要求影像和掩模数据、前景和背景颜色以及光标的 hot point。hot point 表示光标中有效进行点击的位置。对标准的箭头形光标讲，hot point 位于左上角(0,0)，而手形光标取决于如何对它进行绘制，一般在食指下。

```
GdkBitmap *bitmap;
GdkBitmap *mask;
GdkColor white = {0, 0xffff, 0xffff, 0xffff};
GdkColor black = {0, 0x0000, 0x0000, 0x0000};
```

```
/* --- Convert the image data (above) to usable bitmaps --- */
create_bitmap_and_mask_from_xpm (&bitmap, &mask, cursor_hand);

/* --- Create the cursor using the images --- */
cursor = gdk_cursor_new_from_pixmap (
                bitmap, mask, &white, &black, 8, 8);

/* --- Now assign this hand cursor to the window --- */
gdk_window_set_cursor (button->window, cursor);
```

本例用黑和白作为光标颜色,你得到的印象是光标只用两种颜色。事实上光标可以用任何颜色。给 gdk_cursor_new_from_pixmap 函数提供不同的颜色就能做到这一点。虽然可以建立不同颜色的光标,通常最好还是用黑和白两种颜色。

下面是表示 3 个按钮的完整例子。一个用定制的手形作为光标,一个用内置的十字形作为光标,最后一个使用红蓝两色的准星作为光标。最后的例子说明光标可以使用黑和白以外的颜色。

```
/*
 * File: cursors.c
 * Auth: Eric Harlow
 *
 * Illustration of how to change cursors
 */

#include <gtk/gtk.h>

static char * cursor_hand[] = {
"32 32 3 1",
"       c None",
".      c #000000",
"+      c #FFFFFF",
"                                ",
"                                ",
"              ...               ",
"            .++-.   ...         ",
"            .++-.  .+++.   ...  ",
"   ...      .+++.  .+++.  .+++. ",
"  .+++.     .+++. .+-+.  .+++.",
"   .+++.    .+++. .+-+. .+++. ",
"    .+++.   .+++. .+-+. .+++. ",
"     .+++. .+++..+++. .++-.   ",
"      .+++..+++..+++. .++-.   ",
```

```
"        .+++.+++..+++..+++.     ",
"   ...  .++++++++++++++++.      ",
"  .+++. .+++++++-+++++++.       ",
"  .++++..+++++++++++++++.       ",
"   ..++++++++++++++++++.        ",
"     .++++++++++++++++++.       ",
"      .++++++++++++++++.        ",
"       ..++++++++-+++++.        ",
"         .++++++++++++.         ",
"          .++++++++++.          ",
"           ..........           ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                "};

static char * cursor_sight[] = {
"32 32 3 1",
"      c None",
".     c #000000",
"+     c #FF0000",
"                                ",
" .............................  ",
" .            +             .   ",
" .            +             .   ",
" .          +++++           .   ",
" .            +             .   ",
" .       +++++++++++        .   ",
" .            +             .   ",
" .          +++++           .   ",
" .            +             .   ",
" .       +++++++++++        .   ",
" .    +    +    +    +    +  .  ",
" .    +    +  +++++  +    +  .  ",
" .  ++ +  +  +  +  +  +  + + .  ",
" .  ++ +  +  +  +  +  +  + + .  ",
```

```
"  .  ++++++++++++++++++++++++++++.    ",
"  .     ++  +  +  +   +   +  +  ++  +   .    ",
"  .   ++  +  +  +   +   +  +  +  +  +   .    ",
"  .       +   +   +++++   +    +       .    ",
"  .       +   +       +      +    +       .    ",
"  .           ++++++++++             .    ",
"  .                 +                .    ",
"  .               +++++              .    ",
"  .                 +                .    ",
"  .           ++++++++++             .    ",
"  .                 +                .    ",
"  .               +++++              .    ",
"  .                 +                .    ",
"  .                 +                .    ",
"  ..............................     ",
"                                     ",
"                                     ",
"                                     "};

GdkColor white = {0, 0xffff, 0xffff, 0xffff};
GdkColor black = {0, 0x0000, 0x0000, 0x0000};
GdkColor red = {0, 0xffff, 0x0000, 0x0000};
GdkColor blue = {0, 0x0000, 0x0000, 0xffff};


void
create_bitmap_and_mask_from_xpm (GdkBitmap **bitmap, GdkBitmap **mask, gchar **xpm)
{
        int height, width, colors;
        char pixmap_buffer[(32 * 32)/8];
        char mask_buffer[(32 * 32)/8];
        int x, y, pix;
        int transparent_color, black_color;

        sscanf (xpm[0], "%d %d %d %d", &height, &width, &colors, &pix);

        g_assert (height == 32);
        g_assert (width  == 32);
        g_assert (colors == 3);

        transparent_color = ' ';
        black_color = '.';
```

```
        for (y = 0; y < 32; y++){
                for (x = 0; x < 32;){
                        char value = 0, maskv = 0;

                        for (pix = 0; pix < 8; pix++, x++){
                                if (xpm[4+y][x] != transparent_color){
                                        maskv |= 1 << pix;

                                        if (xpm[4+y][x] != black_color){
                                                value |= 1 << pix;
                                        }
                                }
                        }
                        pixmap_buffer[(y * 4 + x/8)-1] = value;
                        mask_buffer[(y * 4 + x/8)-1] = maskv;
                }
        }
        *bitmap = gdk_bitmap_create_from_data (NULL, pixmap_buffer, 32, 32);
        *mask   = gdk_bitmap_create_from_data (NULL, mask_buffer, 32, 32);
}

/*
 * CloseAppWindow
 *
 * Shut down GTK+ when user closes window
 */
gint CloseAppWindow (GtkWidget *widget, gpointer *data)
{
    gtk_main_quit ();
    return (FALSE);
}

/*
 * CreateTable
 *
 * Creates a top-level window with several options.  Because
 * this is going to be done many times, it's best just made
 * into a function.
 */
CreateTable (char  *szTitle, gint xoptions, gint yoptions)
{
```

```
GtkWidget *window;
GtkWidget *button;
GtkWidget *handButton;
GtkWidget *sightButton;
GtkWidget *crossButton;
GtkWidget *table;
GdkBitmap *bitmap;
GdkBitmap *mask;
GdkCursor *cursor;

/* --- Create the top-level window --- */
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

/* --- Give the window a title. --- */
gtk_window_set_title (GTK_WINDOW (window), szTitle);

gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                    GTK_SIGNAL_FUNC (CloseAppWindow), NULL);

/* --- Give the window a border --- */
gtk_container_border_width (GTK_CONTAINER (window), 10);

/* --- Create a 4x4 table --- */
table = gtk_table_new (2, 2, TRUE);

handButton = gtk_button_new_with_label ("Hand");
gtk_table_attach (GTK_TABLE (table), handButton, 0, 1, 0, 1,
                  xoptions, yoptions, 0, 0);
gtk_widget_show (handButton);

crossButton = gtk_button_new_with_label ("Cross");
gtk_table_attach (GTK_TABLE (table), crossButton, 0, 1, 1, 2,
                  xoptions, yoptions, 0, 0);
gtk_widget_show (crossButton);

sightButton = gtk_button_new_with_label ("Gun sight");
gtk_table_attach (GTK_TABLE (table), sightButton, 1, 2, 0, 1,
                  xoptions, yoptions, 0, 0);
gtk_widget_show (sightButton);

/* --- Add the table to the window --- */
gtk_container_add (GTK_CONTAINER (window), table);
```

```
    /* --- Make the table visible --- */
    gtk_widget_show (table);

    /* --- Make the window visible --- */
    gtk_widget_show (window);

    create_bitmap_and_mask_from_xpm (&bitmap, &mask, cursor_hand);
    cursor = gdk_cursor_new_from_pixmap (bitmap, mask, &white, &black, 16, 8);

    gdk_window_set_cursor (handButton->window, cursor);

    create_bitmap_and_mask_from_xpm (&bitmap, &mask, cursor_sight);
    cursor = gdk_cursor_new_from_pixmap (bitmap, mask, &red, &blue, 16, 16);
    gdk_window_set_cursor (sightButton->window, cursor);

    cursor = gdk_cursor_new (GDK_CROSS);
    gdk_window_set_cursor (crossButton->window, cursor);
}


/*
 * --- Main.
 *
 * Program begins here.
 *
 * Rather than duplicate code, the CreateTable function is
 * called.  It does the work for us - all we do is pass in
 * how we want the window to appear.
 */
int main (int argc, char *argv[])
{

    /* --- GTK initialization --- */
    gtk_init (&argc, &argv);

    /* --- No flags set --- */
    CreateTable ("Nothing", 0, 0);

    /* --- Start the gtk loop. --- */
    gtk_main ();
```

```
    exit (0);
}
```

11.3 引用

在 GTK + 中的每件事都包括分配项目。式样可以直接建立或者从已有的式样进行复制,建立构件、装入字体并将它指定给式样,长时间重复这些过程肯定将耗尽你的计算机内存。除非你有方便的办法跟踪它。在 GTK + 中通过对对象的引用(reference)来跟踪内存的使用。

对对象的引用也防止对象被删除。每当取一个对象(如字体),并用 GTK + 子程序将它和其他对象联系起来,这一对象的引用计数就加 1。每次引用的对象被删除,GTK + 取所有它引用的对象,并将这些对象的引用计数减 1。如果对象的引用计数为 0,GTK + 认为这一对象不再需要,并释放对象。

大部分情况是显而易见的,对小型应用讲通常不成问题。如果不小心,比较大的应用可能会出现内存漏损问题。在建立构件时,它的引用计数为 0。当以某种形式将构件指定给容器时,它的引用计数加 1 以表示容器引用了构件。如果删除了容器,容器将它引用的所有构件(容器内的所有东西)的引用计数减 1。如果这些构件中任何一个构件的引用计数为 0,它也将被删除。因为现在不再使用这一构件,这样做是必要的。可以用增加引用计数的办法阻止构件被删除。在这样做的时候,当容器被删除,而构件依然存在。当然,你必须将构件指针保存在某个位置以便以后使用它。否则,这就没有任何意义。

不属于构件的项的处理方式稍稍不同。当建立时,它的引用计数为 1。这些项通常表示式样、像元图或者其他可重复使用的数据。你希望对这些项分配一次,在程序的存在期间重复使用它们。你不希望只是由于最后使用你建立的式样的构件被删除而突然删除式样。然而,对于一次性使用的项通常在使用后撤消引用(dereference)。

```
/* --- Create pixmap from the data --- */
pixmap = gdk_pixmap_create_from_xpm_d (
                main_window->window,
                &mask, NULL, (gchar **) xpm_data);

/* --- Create a pixmap widget --- */
widget = gtk_pixmap_new (pixmap, NULL);

/* --- Don't need the pixmap any more. --- */
gdk_pixmap_unref (pixmap);
```

在这里,撤消对像元图的引用通常导致删除像元图。由 gtk_pixmap_new 函数建立的构件引用像元图。在删除构件时将撤消对像元图的引用。当像元图的引用计数减为 0 时,像元图也将被删除。

11.4　小结

式样由颜色和字体组成，用它来确定构件的外观。可以单独改变构件的式样，也可以对新建立的构件使用默认的式样。GTK + 有一组内置的光标，可以用来改变构件内的光标。你可以方便地建立定制的光标以补充内置的光标。在 GTK + 内，用引用来保持对对象的使用情况的跟踪。当引用计数为 0 时释放对象。

第 12 章　使用 GDK 的分子观察程序

使用 3 维图形显示信息经常是有趣的讨论题目。在本章中，建立一个 .pdb 文件的观察程序在绘图构件中观察分子，并用绘图工具集(GDK)显示分子(见图 12.1)。这一章讲述了使用 3 维数学来显示分子。如果你不熟悉数学，可以忽略矩阵乘法，仍然能从中掌握基本概念。如果你有兴趣学习 3 维数学，有许多讨论 3 维计算机图形的好书。

除了显示分子外，需要增加允许用户旋转分子的程序。在应用中还增加了选项：如使用颜色显示分子和带有原子标号的分子等。为了使分子旋转平稳，这一应用程序使用了双缓冲区技术。

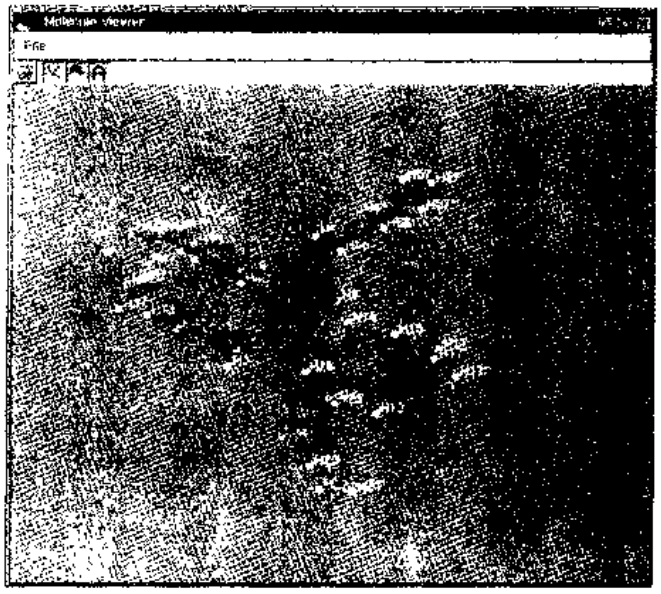


图 12.1　分子观察程序

12.1　文件格式

为了显示 .pdb 文件，必须能够读懂它们和了解描述文件的格式。.pdb 文件的格式很复杂，设计成能够显示复杂的分子。然而，我们感到满意的是可以忽略保存在 .pdb 文件中的大部分信息。唯一有关的项目包括原子的名称、原子的位置以及原子之间的联接。有关的 .pdb 文件分成两个部分：第一部分说明原子，第二部分(可选)说明原子之间的联结。对原子的正式说明非常复杂：

COLUMNS	DATA TYPE	FIELD	DEFINITION
1 - 6	Record name	"ATOM "	
7 - 11	Integer	serial	Atom serial number.
13 - 16	Atom	name	Atom name.
17	Character	altLoc	Alternate location indicator.
18 - 20	Residue name	resName	Residue name.
22	Character	chainID	Chain identifier.
23 - 26	Integer	resSeq	Residue sequence number.
27	AChar	iCode	Code for insertion of residues.
31 - 38	Real(8.3)	x	Orthogonal coordinates for X in Angstroms.
39 - 46	Real(8.3)	y	Orthogonal coordinates for Y in Angstroms.
47 - 54	Real(8.3)	z	Orthogonal coordinates for Z in Angstroms.
55 - 60	Real(6.2)	occupancy	Occupancy.
61 - 66	Real(6.2)	tempFactor	Temperature factor.
73 - 76	LString(4)	segID	Segment identifier, left-justified.
77 - 78	LString(2)	element	Element symbol, right-justified.
79 - 80	LString(2)	charge	Charge on the atom.

分子的重要项目是:"记录名称"、"原子"和原子的 3 维位置(x,y,z)。

.pdb 文件的第二部分是可选的。它包含观察原子联结所需要的信息。其中 CONECT 行说明原子和所有与原子有联结的原子号。在 CONECT 后面的第一个数是第一个原子索引。在同一行中的其他原子与第一个原子之间有联结。带有联结的一般的 .pdb 文件具有类似如下的形式。

```
ATOM      1 O11      1       2.227     3.257     9.904
ATOM      2 O12      1       4.387     2.116    10.202
ATOM      3 O13      1       3.470     2.116     8.123
ATOM      4 N1       1       1.032     3.192    13.498
ATOM      5 C1       1       1.135     4.580    13.046
ATOM      6 C2       1       1.192     2.116    12.739
ATOM      7 C3       1       0.762     2.796    14.797
ATOM      8 C3'      1       0.507     3.783    15.877
ATOM      9 C11      1       2.759     4.526     9.626
ATOM     10 C12      1       4.372     2.116    11.603
ATOM     11 C13      1       2.451     2.116     7.171
ATOM     12 B1       1       3.078     2.116     9.530
ATOM     13 N1 *     1       1.032     1.041    13.498
ATOM     14 C3 *     1       0.762     1.437    14.797
```

```
ATOM     15 O11 *          1       2.227    0.976     9.904
ATOM     16 C1 *           1       1.135   -0.347    13.046
ATOM     17 C3' *          1       0.507    0.450    15.877
ATOM     18 C11 *          1       2.759   -0.293     9.626
TER
CONECT    1    9   12
CONECT    2   10   12
CONECT    3   11   12
CONECT    4    5    6    7
CONECT    5    4
CONECT    6    4   13
CONECT    7    4    8   14
CONECT    8    7
CONECT    9    1
CONECT   10    2
CONECT   11    3
CONECT   12    1    2    3   15
CONECT   13    6   14   16
CONECT   14    7   13   17
CONECT   15   12
CONECT   16   13
CONECT   17   14
```

现在来看文件中的第一行。行以 ATOM 开始,表示这是说明原子的行。下一个字符"1"是原子的索引。原子必须是顺序的。O11 是原子的名称。在这里,O 可能代表氧。原子的位置用行中最后 3 个数字表示。它们应该是原子在结构中的 x、y 和 z 位置。

```
ATOM    1  O11       1        2.227    3.257    9.904
```

说明原子的行最后以带标号 TER 的行作为结束,接着是说明联结的行。联结用 CONECT 字符串作为开始,包含原子号和与它有联结的原子。例如,联结的第一行如下:

```
CONECT    1    9   12
```

这一行表示 1 号原子应该与 9 号和 12 号原子联结。原子之间不一定需要有联结,但它们之间通常有联结。

12.2 数据结构

用于原子的数据结构(typAtom)保存原子的名称,两组坐标以及与它有联结的原子列表。原子结构有两组坐标,因为它保存了原始坐标和"转换"后的坐标。因为对象将围绕坐标轴旋转,保存转换坐标是需要的。随着对象的旋转,原子将出现在 3 维空间的不同位置上。新的位置也就是转换后的位置,每次移动分子时都要对原子的位置进行计算。应该在转换后的位置

上显示原子。由于每个原子都与一个或多个其他原子相联结，下面的 typAtom 结构在 GSList 中存储这些联结。

```
typedef struct {
    char * szName;          /* --- Molecule name --- */
    double x;               /* --- Original coordinates --- */
    double y;               /*
    double z;               /*
    double tx;              /* --- Translated coordinates --- */
    double ty;              /*
    double tz;              /*
    GSList * bondList;      /* --- Atoms it has a bond with --- */
} typAtom;
```

联结反映两个原子之间的关系，应该在它们之间画联接线。此外，用一个标志来加速绘制过程。一对原子之间的联结应该只画一次。

```
typedef struct {
    typAtom * atom1;    /* --- First atom in the bond --- */
    typAtom * atom2;    /* --- Second atom in the bond --- */
    int bPainted;       /* --- painted --- */
} typBond;
```

原子和联结都被保存在静态数组中。可以将它们以更为动态的方式保存，但需要花费更多的工作量。在绘制时，数组使原子易于处理。

12.3　3 维绘图

由于分子是 3 维结构，应该将分子画成具有 3 维的幻觉。在绘制分子时不要犯极明显的错误。明显的错误是在绘制分子时不管它们的视觉 z 深度。这种情况可能导致离观察者远的原子出现在离观察者近的原子的上面。解决这个问题的办法是在它们转换到新的位置后按转换后的 z 深度对原子进行分类。先画最远的原子，保证距离近的原子不会被距离远的原子所覆盖。“绘图者”算法将先画距离远的原子，并用距离近的原子覆盖它们。

12.4　源代码

所有处理分子的主要子程序(矩阵运算程序除外)都放在 molecule.c 文件中。读数据、绘制工作和其他的分子处理都包含在这个文件中。将程序以这种方式模块化，可以很容易将模块放在其他应用程序中。由于大多数新的代码放在 molecule.c 文件中，我们可以浏览主要的模块，并依次讨论它们的功能。

12.4.1 ReadMolecule

主要的用于装入文件的子程序分成两个部分。首先,从文件读入原子的名称和坐标。然后读入联结,并将它们加到原子数据结构和联结数组中去。当应用程序启动时调用 ReadMolecule,默认地设置为读取已经存在的分子。也可以通过 File 菜单的 Open 选项调用 ReadMolecule,它允许读入其他的 .pdb 文件。当装入新文件时,需要对一些数据(如矩阵)进行清除。在读入分子数据后,ReadMolecule 子程序强制实施重新绘制工作。

```
/*
 * ReadMolecule
 *
 * Read in a .pdb molecule with the given filename.
 * Store the information in the data structures
 * defined.
 */
void ReadMolecule (char *sFilename)
{
    FILE *fp;
    char buffer[120];
    float x, y, z;
    int nIndex1;
    int nIndex2;
    char szName[120];
    char *sTmp;
    typAtom *atom;
    char szTmp[20];

    Init3d();

    nAtoms = 0;
    nBonds = 0;

    /* --- Reset matrices if reading in new file --- */
    if (mat) {
        unit (mat);
        unit (amat);
        unit (tmat);
    }

    nMoleculeRadius = 2;
```

```
/* --- Open the file for reading --- */
fp = fopen (sFilename, "r");

/* --- Read in a line from the file --- */
while (fgets (buffer, sizeof (buffer), fp)) {

    /* --- Why it's an 'atom' --- */
    if (strncmp (buffer, "ATOM", 4) == 0) {

        /* --- Read in the atom based on the known file
         *       structure
         */
        strncpy (szName, &buffer[12], 4);
        szName[4] = 0;
        strncpy (szTmp, &buffer[30], 8);
        szTmp[8] = 0;
        x = atof (szTmp);
        strncpy (szTmp, &buffer[38], 8);
        szTmp[8] = 0;
        y = atof (szTmp);
        strncpy (szTmp, &buffer[46], 8);
        szTmp[8] = 0;
        z = atof (szTmp);

        /* --- Atoms are base 1 - index first. --- */
        nAtoms++;

        /* --- Populate the data structure -     */
        atom = &atomlist[nAtoms];
        atom->x = x;
        atom->y = y;
        atom->z = z;
        atom->szName = strdup (szName);
        atom->bondList = NULL;
        sortindex[nAtoms-1] = nAtoms;

    /* --- Does this describe a bond? --- */
    } else if (strncmp (buffer, "CONECT", 6) == 0) {

        /* --- Get the first bond atom --- */
        sTmp = GetNextValue (&buffer[6], &nIndex1);
```

```
            /* --- Get the next bond atom --- */
            while (sTmp = GetNextValue(sTmp, &nIndex2)) {

                /* --- This bond is this nIndex1 to nIndex2 --- */
                bondlist[nBonds].atom1 = &atomlist[nIndex1];
                bondlist[nBonds].atom2 = &atomlist[nIndex2];

                /* --- Of course the atom should know which
                 *      bonds it is a part of.
                 */
                atomlist[nIndex1].bondList =
                   g_slist_append(atomlist[nIndex1].bondList,
                                  &bondlist[nBonds]);

                /* --- Both atoms should know... --- */
                atomList[nIndex2].bondList =
                   g_slist_append(atomlist[nIndex2].bondList,
                                  &bondlist[nBonds]);

                /* --- Just another bond... James Bond. --- */
                nBonds++;
            }
        }
    }

    /* --- Find bounding box --- */
    FindBB();

    /* --- Sort the molecules --- */
    SortMolecules(atomlist, sortindex);

    MoleculeRepaint();
}
```

12.4.2 FindBB

有界框(bounding box)是适合显示分子结构的 3 维框。FindBB 子程序计算有界框的大小,并允许缩放对象使它适合屏幕显示。

```
/*
 * FindBB
 *
```

```
 * Find the minimum bounding box that would fit all the
 * atoms in the molecule
 */
void FindBB ()
{
    int         i;
    typAtom     * atom;

    /* --- At first, the box consisted of a single atom --- */
    /* --- Atoms start at 1 --- */
    atom = &atomlist[1];

    xmin = atom->x;
    xmax = atom->x;

    ymin = atom->y;
    ymax = atom->y;

    zmin = atom->z;
    zmax = atom->z;

    /* --- Then we added a whole bunch of them --- */
    for (i = 2; i <= nAtoms; i++) {

        atom = &atomlist[i];
        if (atom->x < xmin) xmin = atom->x;
        if (atom->x > xmax) xmax = atom->x;
        if (atom->y < ymin) ymin = atom->y;
        if (atom->y > ymax) ymax = atom->y;
        if (atom->z < zmin) zmin = atom->z;
        if (atom->z > zmax) zmax = atom->z;
    }

    /* --- And now we have our bounding box. --- */
}
```

12.4.3　对原子进行分类

为了得到好的 3 维影像，绘制分子结构需要从最远的原子画到最近的原子。必须根据转换后的 z 坐标对原子进行分类。不能使用原始的 z 位置，因为用户可能旋转分子，而且观察者看到的是旋转后的分子。在分类后，最后画比较近的原子，而且它出现在最前面。代替对原子本身进行分类，实际对原子的索引进行分类。在数组中移动整数远比在内存中移动结构快。

```
/*
 * SortMolecules
 *
 * Call the sorting function
 */
void SortMolecules (typAtom * atomlist, int * sortindex)
{
    QSortMolecules (atomlist, sortindex, 0, nAtoms-1);
}

/*
 * QSortMolecules
 *
 * This is just a quicksort of the atoms in the molecule.
 *
 * Note: Rather than sort the atomlist itself, the sortindex
 * is the item being sorted.    Manipulating integers is just
 * quicker than moving structures around in memory - especially
 * because we have to do this in real time over and over again.
 *
 * atomlist - list of atoms to sort
 * sortlist - index of indexes.
 */
void QSortMolecules (typAtom * atomlist, int * sortindex, int lo0, int hi0)
{
    int    nTmp;
    int    lo = lo0;
    int    hi = hi0;
    int    mid;

    if (hi0 > lo0) {

        mid = atomlist[sortindex[(lo0 + hi0) / 2]].tz;

        while (lo <= hi) {

            while (lo < hi0 && atomlist[sortindex[lo]].tz < mid) {
                lo++;
            }

            while (hi > lo0 && atomlist[sortindex[hi]].tz > mid) {
                hi--;
```

```
            }

            if (lo < = hi) {

                nTmp = sortindex[lo];
                sortindex[lo] = sortindex[hi];
                sortindex[hi] = nTmp;
                lo++;
                hi--;
            }
        }
        if (lo0 < hi) QSortMolecules (atomlist, sortindex, lo0, hi);
        if (lo < hi0) QSortMolecules (atomlist, sortindex, lo, hi0);
    }
}
```

12.4.4 TransformPoints

TransformPoints 子程序对分子中的所有原子使用矩阵运算。这一操作具有旋转影像的效果，并缩放影像以适合在屏幕上显示它。实际的矩阵运算没有列在下面。

```
/*
 * TransformPoints
 *
 * Take the atoms and rotate them to their displayed
 * position so they can be drawn.
 *
 * mat - The matrix used to calculate the new atom position
 */
void TransformPoints (typMatrix3D *mat)
{
    int    i;

    for (i = 1; i < = nAtoms; i++) {

        Transform (mat, &atomlist [i]);
    }
}
```

paint

paint 子程序本身实际不进行绘图，但由需要绘图的程序调用它。实际绘图工作由 Draw-

Molecule 子程序完成。但在调用 DrawMolecule 以前需要完成使对象旋转的矩阵运算,并对每个原子完成相应的运算。然后才能够画分子的结构。

```
/*
 * paint
 *
 * paint the molecule on the screen, well, at
 * least to the background.    Most of this routine
 * works on getting the matrices correct before calling
 * the routine that does the real painting.
 */
void paint (typGraphics *g)
{
    double xfac;
    double f1;
    double f2;

    /* --- What's the range (delta x, delta y, delta z) --- */
    double xw = xmax - xmin;
    double yw = ymax - ymin;
    double zw = zmax - zmin;

    /* --- Make sure everything is allocated --- */
    Init3d();

    /* --- Calculate the factor to scale the molecule --- */
    if (yw > xw) xw = yw;
    if (zw > xw) xw = zw;
    f1 = nScreenWidth / xw;
    f2 = nScreenHeight / xw;
    xfac = .7 * (f1 < f2 ? f1 : f2);

    /* --- First make the matrix the unit matrix --- */
    unit (mat);

    /* --- Translate it around the origin.    This moves the
     *      molecule to be centered around the axis as we're
     *      moving it based on the half of the bounding box width.
     */
    translate (mat, -(xmin + xmax) / 2,
                    -(ymin + ymax) / 2,
                    -(zmin + zmax) / 2);
```

```
    /* --- Rotate the image around the axis.     amat is the
     *      matrix that represents how much to rotate the
     *      molecule.
     */
    mult (mat, amat);

    /* --- Scale the molecule based on the screen width --- */
    scale3 (mat, xfac, -xfac, 16 * xfac / nScreenWidth);

    /* --- Translate molecule based on screen width and height --- */
    translate (mat, nScreenWidth / 2, nScreenHeight / 2, 10);

    /* --- Calculate the new position of all the points --- */
    TransformPoints (mat);

    /* --- Draw the molecule based on translated coordinates --- */
    DrawMolecule (g);
}
```

12.4.5　绘制联结

在绘制每个原子以后，就可以绘制原子之间的联结。因为哪些原子之间具有联结是已知的。然而，这种方法使每一联结重复画两次，这是不必要的。你可以在画原子以前或以后画联结，但两种情况都可能损害希望达到的 3 维效果。好的做法是在一对联结中从距离远的原子开始画联结。这种方法只画一次，而且联结不覆盖离观察者近的原子。

使用 DrawMolecule 函数绘制分子。用经过分类的 sortindex 数组绘制原子。使用 sortindex 数组使得绘制工作根据离观察者的距离来进行。用 bPainted 标志保持对已经绘制的联结的跟踪。在绘制一条联结后，对这一帧影像就不再重复画第二次(没有必要浪费 CPU 时间)。沿着有联系的原子绘制联结，以达到理想的 3 维效果。

```
/*
 * DrawMolecule
 *
 * Draw the molecule.     'nuff said.
 */
void DrawMolecule (typGraphics *g)
{
    int nIndex;
    int nDiameter;
    typBond *bond;
    GSList *list;
```

```
typAtom * atom;
int i;
GdkGC * pen;

/* --- Make sure everything's ready --- */
Init3d ();

/* --- Sort the molecules --- */
SortMolecules (atomlist, sortindex);

/* --- Get the molecule diameter --- */
nDiameter = nMoleculeRadius + nMoleculeRadius;

/* --- If we're showing bonds --- */
if (ShowLines()) {

    /* --- Clear out the flag that indicates that this
     *      particular bond has been painted.
     */
    for (i = 0; i < nBonds; i++) {

        bondlist[i].bPainted = FALSE;
    }
}

/* --- Display all the atoms in the list --- */
for (i = 0; i < nAtoms; i++) {

    /* --- Use the sort list - draw from farthest back
     *      moving up.
     */
    nIndex = sortindex[i];

    /* --- Get the atom the index refers to --- */
    atom = &atomlist[nIndex];

    /* --- This atom has a color --- */
    pen = GetAtomColor (atom);

    /* --- Draw a circle with the atom color --- */
    gdk_draw_arc (g->pixmap, pen, TRUE,
                  atom->tx - 3, atom->ty - 3, 7, 7, 0, 360 * 64);
```

```
        /* --- If they want labels --- */
        if (ShowLabels ()) {

            /* --- Show the label --- */
            if (atom->szName) {
                gdk_draw_string (g->pixmap, font,
                                 pen, atom->tx + 5, atom->ty,
                                 atom->szName);
            }
        }

        /* --- If they want lines (bonds) --- */
        if (ShowLines()) {

            /* --- Show all bonds for this atom --- */
            for (list = atom->bondList; list; list = list->next) {

                /* --- Get the bond from the list --- */
                bond = (typBond *) list->data;

                /* --- If it hasn't been painted yet --- */
                if (bond->bPainted == FALSE) {

                  /* --- Paint the bond (draw the line) --- */
                  DrawBond (g, bond->atom1, bond->atom2);

                  /* --- Bond marked as painted --- */
                  bond->bPainted = TRUE;
                }
            }
        }
    }
}
```

DrawBond 函数用单线联接两个原子。联线是用转换后的坐标绘制的。

```
/*
 * DrawBond
 *
 * Draw a bond between these two atoms.
 *
```

```
 * g - our data structure that has information used to draw
 * atom1 - first atom in the bond pair
 * atom2 - second atom in the bond pair.
 */
void DrawBond (typGraphics *g, typAtom *atom1, typAtom *atom2)
{
    gdk_draw_line (g->pixmap, penBlack,
                   (int) (atom1->tx),
                   (int) (atom1->ty),
                   (int) (atom2->tx),
                   (int) (atom2->ty));

}
```

12.4.6 原子的颜色

如果选择了彩色工具条按钮，将以彩色显示原子。如果没有选择彩色工具条，则以黑色显示原子。如果按下了彩色工具条按钮，用彩色显示原子。使用的具体颜色取决于在文件中的原子名称。例如，如果原子的名称以 n 开始，表示这可能是氮原子，则返回带兰色笔的 GdkGC。

```
/*
 * GetAtomColor
 *
 * Gets the color of the atom based on some of the
 * characteristics of the atom.
 *
 * The molecules are either drawn black or drawn in
 * color depending on a flag set by a toolbar button.
 *
 * If the flag is set...
 * If the name begins with a 'C' (carbon) then we draw
 * it in black.    If the name begins with 'N' (nitrogen)
 * then we draw it in blue.    If the name begins with 'O'
 * (oxygen) then we draw in in red.    Everything else is
 * drawn in gray.
 *
 * atom - atom to have its color chosen
 */
GdkGC *GetAtomColor (typAtom *atom)
{
    char szName[10];
```

```
    /* --- Don't know the name. --- */
    if (atom->szName == NULL) {
        return (penBlack);
    }

    /* --- Don't show colors --- */
    if (!ShowRGB()) {
        return (penBlack);
    }

    GetAtomName (szName, atom);

    if (!strcmp (szName, "CL")) {
        return (penGreen);
    } else if (!strcmp (szName, "C")) {
        return (penBlack);
    } else if (!strcmp (szName, "S")) {
        return (penYellow);
    } else if (!strcmp (szName, "P")) {
        return (penOrange);
    } else if (!strcmp (szName, "N")) {
        return (penBlue);
    } else if (!strcmp (szName, "O")) {
        return (penRed);
    } else if (!strcmp (szName, "F")) {
        return (penWhite);
    } else {
        return (penDarkGray);
    }
```

12.4.6.1　configure_event

configure_event 子程序对结构进行初始化。由于在建立构件时，或者在重新改变构件大小时调用这一子程序，configure_event 按构件的大小分配背景像元图。

```
/*
 * configure_event
 *
 * Create a new backing pixmap of the appropriate size
 */
```

```
static gint configure_event (GtkWidget *widget, GdkEventConfigure *event)
{
    /* --- Structure doesn't exist? --- */
    if (g == NULL) {

        /* --- Create one --- */
        g = NewGraphics ();
    }

    /* --- Existing pixmap?      --- */
    if (g->pixmap) {

        /* --- Free it --- */
        gdk_pixmap_unref (g->pixmap);
    }

    /* --- Create a new pixmap --- */
    g->pixmap = gdk_pixmap_new (widget->window,
                                widget->allocation.width,
                                widget->allocation.height,
                                -1);

    /* --- Get height and width to clear screen --- */
    nScreenWidth = widget->allocation.width;
    nScreenHeight = widget->allocation.height;

    /* --- Create a new pixmap --- */
    gdk_draw_rectangle (g->pixmap,
                        widget->style->white_gc,
                        TRUE,
                        0, 0,
                        widget->allocation.width,
                        widget->allocation.height);

    /* --- Redraw molecule --- */
    MoleculeRepaint ();

    return TRUE;
}
```

12.4.6.2 expose_event

在屏幕上的一个区域需要进行更改时调用 expose_event 子程序。对这一应用讲，expose

_event 子程序只将背景像元图复制到可绘制构件上。因此，expose _ event 子程序执行得很快。

```
/*
 * expose _ event
 *
 * Called when an area on the screen has been exposed
 * and we have to repaint it.    The area is repainted from
 * the background area we have.
 */
static gint expose _ event (GtkWidget *widget, GdkEventExpose *event)
{

    gdk _ draw _ pixmap(widget->window,
            widget->style->fg _ gc[GTK _ WIDGET _ STATE (widget)],
            g->pixmap,
            event->area.x, event->area.y,
            event->area.x, event->area.y,
            event->area.width, event->area.height);

    return FALSE;
}
```

12.4.7　MoleculeRepaint

当要求重新绘制分子结构时调用 MoleculeRepaint 函数。它清除背景像元图，调用 expose _ event 子程序将背景像元图复制到窗口，绘制分子结构。

```
/*
 * MoleculeRepaint
 *
 * Called when the user uses the mouse to move the
 * molecule.    This causes the molecule to be repainted on
 * the screen.
 */
void MoleculeRepaint ()
{
    GdkRectangle    update _ rect;

    Init3d ();

    /* --- clear pixmap --- */
    gdk _ draw _ rectangle (g->pixmap,
```

```
                    penGray,
                    TRUE,
                    0, 0,
                    drawing_area->allocation.width,
                    drawing_area->allocation.height);

    /* --- Draw molecule in the background --- */
    paint (g);

    /* --- The whole screen --- */
    update_rect.x = 0;
    update_rect.y = 0;
    update_rect.width = drawing_area->allocation.width;
    update_rect.height = drawing_area->allocation.height;

    /* --- Call the expose event - which copies the background
     *     it to the widget
     */
    gtk_widget_draw (drawing_area, &update_rect);
}
```

每当用户用鼠标操作图形时,重新将分子结构画在背景像元图上。这个程序和第 10 章"绘图工具集"中的双缓冲区时钟应用程序的唯一区别在于:在时钟应用中,定时器导致每秒重新绘制构件一次;在这个程序中,用户拉动鼠标导致在背景像元图上重新绘制分子结构。

12.4.8 建立绘图区

建立应用的 frontend.c 程序调用 CreateDrawingArea 函数。这一段程序封装建立分子绘图区所需要的一切,但不涉及应用的具体细节。可以不加修改地将这个函数插入到任何程序中。绘图区构件监听 expose_event 信号;所以,它可以重新绘制窗口。绘图区构件同时监听 configure_event 信号,所以,它可以对背景像元图的尺寸进行必要的配置,并作好准备在背景像元图上进行绘制。还需要监听 motion_notify_event 信号,使你知道什么时候用户用鼠标拉动分子结构。然而,只用 gtk_signal_connect 函数连接 motion_notify_event 有问题,在 GTK+中,并不将低层的 GDK 事件传送给应用程序,除非你明确告诉 GTK+传送这些事件。

可以使用 gtk_widget_set_events 函数来做这件事情。调用函数时应该传递下列值之一作为参数:

- GDK_EXPOSURE_MASK
- GDK_POINTER_MOTION_MASK
- GDK_POINTER_MOTION_HINT_MASK
- GDK_BUTTON_MOTION_MASK
- GDK_BUTTON1_MOTION_MASK

- GDK_BUTTON2_MOTION_MASK
- GDK_BUTTON3_MOTION_MASK
- GDK_BUTTON_PRESS_MASK
- GDK_BUTTON_RELEASE_MASK
- GDK_KEY_PRESS_MASK
- GDK_KEY_RELEASE_MASK
- GDK_ENTER_NOTIFY_MASK
- GDK_LEAVE_NOTIFY_MASK
- GDK_FOCUS_CHANGE_MASK
- GDK_STRUCTURE_MASK
- GDK_PROPERTY_CHANGE_MASK
- GDK_PROXIMITY_IN_MASK
- GDK_PROXIMITY_OUT_MASK

对这一应用讲，需要使用 GTK_POINTER_MOTION_MASK 值，在鼠标移动时通知你。但仍有一个小问题：当用户移动鼠标时鼠标事件经常发生。如果机器响应鼠标事件的速度不够快，或者处理时间过长，需要将鼠标事件排队进行等待。例如，如果你想画一个很大的分子，有大量的计算和绘图工作量，就有可能出现上述情况。在这种情况下，即使用户停止移动鼠标，计算和绘图工作仍将继续一段时间（清除事件队列），因而不可能显示清晰的图形（为了在慢速的机器上看到这种情况，使用 GTK_POINTER_MOTION_MASK 值，快速移动带有颜色、联结和标号显示的复杂的分子，几秒钟后再停下来）。好的做法是利用鼠标提示（hint）。

通常每次移动鼠标都产生一个事件，并排队等候处理。当应用程序努力处理鼠标移动事件队列时，将导致应用程序出现响应变慢的现象。好的做法是利用 GTK_POINTER_MOTION_HINT_MASK，它允许应用程序按它自己的步调去响应鼠标事件。当鼠标移动时，鼠标事件并不排队等候。作为代替，只有一个鼠标事件排队以表示鼠标移动了。由于设置了提示标志，在事件中不提供鼠标具体在哪里移动的信息。作为代替，应用程序应该使用 gdk_widget_get_pointer 函数查询鼠标的当前位置。

使用通常的鼠标移动事件和使用提示之间的差异可以从简单的例子中看到。假定你写一个应用程序（例如，分子观察程序），用它绘制在原子上带有 3 维阴影的更复杂的模型。它在你的一流的 600MHz 的双处理器机器上工作很出色。你决定在你的 Web 页上将应用程序提供公众使用。一个拥有 16 MHz 的 386SX 的用户下载了这个应用程序，希望用它来分析复杂的模型。现在，假定你在写应用时使用了标准的鼠标移动事件。在你的一流机器上，你用鼠标旋转模型时大约需要用百分之一秒的时间来绘制它。当你用鼠标以这种或那种方式拉动模型时一切工作正常。然而，在 386 上，绘制模型要用 20 秒的时间。如果机器在吃力地绘制分子结构时用户试图拉动模型，他会感到奇怪：为什么分子结构老是没有画完。20 秒后机器画完了分子结构，又会出现什么情况呢？在此期间，在队列中可能有几百个事件在排队。即使用户以后不做任何事，每一个事件都将导致重新绘制分子，将消耗机器很长一段时间。然而，如果应用程序使用鼠标提示代替鼠标移动事件，当绘制工作完成时，在队列中只有一个表示鼠标移动的事件。查询鼠标当前所处的位置，一切处理也就完成了。当然，这个例子是极端的。谁也不

会在 386 上进行 3 维制图。

其次是检测用户按下或释放鼠标按钮。相应地需要设置 GDK _ BUTTON _ PRESS _ MASK 和 GDK _ BUTTON _ RELEASE _ MASK 标志。必须在构件实现之前改变取得低层事件的事件掩码(mask)。最好在建立构件后立即设置掩码。下面的例子说明如何建立和配置绘图区,使它接受鼠标移动事件。

```
GtkWidget  * CreateDrawingArea ()
{
    GtkWidget * window;

    /* --- Create a top-level window --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- Create the drawing area      --- */
    drawing_area = gtk_drawing_area_new ();

    /* --- Set the size --- */
    gtk_drawing_area_size (GTK_DRAWING_AREA (drawing_area), 300, 300);

    /* --- Make drawing area visible --- */
    gtk_widget_show (drawing_area);

    /* --- Signals used to handle background pixmap --- */
    gtk_signal_connect (GTK_OBJECT (drawing_area), "expose_event",
                        (GtkSignalFunc) expose_event, NULL);
    gtk_signal_connect (GTK_OBJECT(drawing_area),"configure_event",
                        (GtkSignalFunc) configure_event, NULL);

    /* --- Need to know about mouse movements. --- */
    gtk_signal_connect (GTK_OBJECT (drawing_area),"motion_notify_event",
                        (GtkSignalFunc) motion_notify, NULL);

    /* --- Events to listen for --- */
    gtk_widget_set_events (drawing_area, GDK_EXPOSURE_MASK
                           | GDK_LEAVE_NOTIFY_MASK
                           | GDK_BUTTON_PRESS_MASK
                           | GDK_POINTER_MOTION_MASK
                           | GDK_POINTER_MOTION_HINT_MASK);

    return (drawing_area);
}
```

为了看到使用鼠标提示所产生的差异，可以在 gtk _ widget _ set _ events 函数调用中去掉 GDK _ POINTER _ MOTION _ HINT _ MASK。当然，你必须使用非常复杂的分子或者一台很慢的机器才能看到鼠标事件排队的情况。

12.4.8.1 motion _ notify

motion _ notify 函数检测鼠标移动，并提取需要绘制的分子。调用 mouseDrag 函数旋转分子。用 prevx 和 prevy 位置跟踪鼠标原先的位置。鼠标原先的位置用来计算鼠标拉动的方向。知道鼠标拉动的方向，你可以用这些信息建立矩阵和旋转分子。当然，这一函数同时处理鼠标提示和通常的鼠标移动事件。如果绘图区设置了鼠标提示，则 event->is _ hint 将为真。在每次调用 motion _ notify 后查询鼠标位置。

```
/*
 * motion_notify
 *
 * Called when the user moves the mouse on the screen
 */
gint motion_notify (GtkWidget *widget, GdkEventMotion *event)
{
    int x, y;
    GdkModifierType state;

    /* --- If it's a hint... (combining several events) --- */
    if (event->is_hint) {

        /* --- Get new position --- */
        gdk_window_get_pointer (event->window, &x, &y, &state);
    } else {

        /* --- Get new position --- */
        x = event->x;
        y = event->y;
        state = event->state;
    }

    /* --- If the mouse button is down --- */
    if (state & GDK_BUTTON1_MASK && g->pixmap != NULL) {

        /* --- Calculate the mouse drag effect on the
         *     molecule.
         */
        mouseDrag (x, y);
```

```
    }

    /* --- Keep track of the position of the mouse --- */
    prevx = x;
    prevy = y;

    return TRUE;
}
```

12.4.8.2 mouseDrag

mouseDrag 函数重新计算旋转分子的矩阵。回忆一下,每个原子有一个从 .pdb 文件读入的位置(x, y, z)和一个用来画分子的转换后的位置。两者的关系是:将原始的原子位置乘以原子矩阵(amat)旋转分子以得到新的转换位置。调整原子矩阵以反映鼠标的移动。建立新的矩阵(tmat),并调整它使它反映鼠标在 x 和 y 轴方向的拉动。然后将新矩阵和原子矩阵相乘,使原子矩阵反映鼠标拉动的变化。将原始坐标与原子矩阵相乘产生转换后的原子位置。然后调用 MoleculeRepaint 函数利用通过原子矩阵得到的分子的新位置在屏幕上重新绘制分子。

```
/*
 * mouseDrag
 *
 * Calculate the new rotation of the molecule based on how
 * the user drags the mouse over the molecule.
 *
 * x - x position of mouse
 * x - y position of mouse
 */
int mouseDrag (int x, int y)
{

    /* --- Calculate the x difference --- */
    double xtheta = (prevy - y) * (360.0f / nScreenWidth);

    /* --- Calculate the y difference --- */
    double ytheta = (x - prevx) * (360.0f / nScreenHeight);

    /* --- Unit matrix --- */
    unit (tmat);

    /* --- Rotate by the -x- the mouse moved --- */
    xrot (tmat, xtheta);
```

```
    /* --- Rotate by the -y- the mouse moved --- */
    yrot (tmat, ytheta);

    /* --- Combine into existing rotation to get new rotation --- */
    mult (amat, tmat);

    /* --- Redraw with new rotation --- */
    MoleculeRepaint ();

    return TRUE;
}
```

12.4.8.3　frontend.c

frontend.c 程序具有建立主屏幕、建立菜单和工具条以及处理主要事件(工具条按钮,菜单等)的函数。这些程序几乎和在前面的模块中的一样。一些改变使得分子观察程序的菜单和工具条有它自己的特色。

```
/*
 * File: frontend.c
 * Auth: Eric Harlow
 *
 * GUI front end to the molecule viewer.
 *
 */

#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <gtk/gtk.h>

/* ---------------------------- */
/* --- Function prototypes --- */
/* ---------------------------- */
void MoleculeRepaint ();
void MoleculeShowLines (int bValue);
void ReadMolecule (char *);
GtkWidget *CreateDrawingArea ();
static void CreateMainWindow ();
void CreateToolbar (GtkWidget *vbox_main);
void SetToolbarButton (char *szButton, int nState);
void SelectMenu (GtkWidget *widget, gpointer data);
```

```
void DeSelectMenu (GtkWidget * widget, gpointer data);
void SetMenuButton (char * szButton, int nState) ;
GtkWidget * CreateWidgetFromXpm (GtkWidget * window, gchar ** xpm_data);
GtkWidget * CreateMenuItem (GtkWidget * menu,
                            char * szName,
                            char * szAccel,
                            char * szTip,
                            GtkSignalFunc func,
                            gpointer data);
GtkWidget * CreateMenuCheck (GtkWidget * menu,
                             char * szName,
                             GtkSignalFunc func,
                             gpointer data);
GtkWidget * CreateSubMenu (GtkWidget * menubar, char * szName);
GtkWidget * CreateBarSubMenu (GtkWidget * menu, char * szName);


/* ------------------------- */
/* --- Global variables --- */
/* ------------------------- */
GtkWidget            * win_main;
GtkTooltips          * tooltips;
GtkAcceleratorTable * accelerator_table;
GtkWidget            * toolbar;
GtkWidget            * tool_lines;
GtkWidget            * tool_rgb;
GtkWidget            * tool_labels;


/*
 * --- Bitmap for "open"
 */
static const gchar * xpm_open[] = {
"16 16 4 1",
"  c None",
"B c #000000000000",
"Y c #FFFFFFFF0000",
"y c #999999990000",
"                ",
"      BBB       ",
"  BBBB  B   BB  ",
"  BYYYB     BB  ",
```

```
" BYYYYYBBBBB      ",
" BYYYYYYYYB       ",
" BYYYYYYYYB       ",
" BYYYYYYYYB       ",
" BYYBBBBBBBBBB    ",
" BYYByyyyyyyyB    ",
" BYByyyyyyyyB     ",
" BYByyyyyyyyB     ",
" BByyyyyyyyB      ",
" BByyyyyyyyB      ",
" BBBBBBBBBB       ",
"                  ",
};

/*
 * --- Bitmap for "lines"
 */
static const char *xpm_lines[] = {
"16 16 2 1",
"  c None",
"B c #000000000000",
"                  ",
"                  ",
"  BB        BB    ",
"  BB        BB    ",
"   B        B     ",
"   B       B      ",
"    B     B       ",
"    B   B         ",
"     B B          ",
"      BB          ",
"      BBBB        ",
"          BB      ",
"            BBB   ",
"              BB  ",
"                  ",
"                  ",
};

/*
 * --- Bitmap for "color"
 */
```

```
static const char * xpm_rgb[] = {
"16 16 4 1",
"     c None",
"R    c #FF0000",
"G    c #00FF00",
"B    c #0000FF",
"                ",
"      BBBRRR    ",
"    BBBBBRRRRR  ",
"   BBBBBBRRRRRR ",
"  BBBBBBRRRRRRR ",
"  BBBBBBRRRRRRR ",
" BBBBBBBRRRRRRRR ",
" BBBBBBBRRRRRRRR ",
" BBBBBGGGGRRRRR ",
" BBBGGGGGGGGRRR ",
"  GGGGGGGGGGG   ",
"  GGGGGGGGGGG   ",
"  GGGGGGGGGGG   ",
"     GGGGGGGGG  ",
"       GGGGGG   ",
"                ",
};


/*
 * --- Bitmap for "Labels"
 */
static const char * xpm_labels[] = {
"16 16 4 1",
"   c None",
"R c #FF0000",
"G c #00FF00",
"B c #000000",
"                ",
"       BB       ",
"     BBBBBB     ",
"    BBB  BBB    ",
"   BB      BB   ",
"   BB      BB   ",
"   BB      BB   ",
"   BB      BB   ",
```

```
"    BBBBBBBBBBBB    ",
"    BBBBBBBBBBBB    ",
"    BB        BB    ",
"    BB        BB    ",
"    BB        BB    ",
"    BB        BB    ",
"    BB        BB    ",
"                    ",
};


/*
 * EndProgram
 *
 * xit from the program
 */
void EndProgram()
{
    gtk_main_quit();
}


/*
 * main
 *
 * --- Program begins here
 */
int main(int argc, char *argv[])
{
    /* --- Initialize GTK+ --- */
    gtk_init(&argc, &argv);

    /* --- Initialize tooltips --- */
    tooltips = gtk_tooltips_new();

    /* --- Create window(s) --- */
    CreateMainWindow();

    /* --- Read in the default molecule --- */
    ReadMolecule("molecule.pdb");

    /* --- Main event handling loop --- */
```

```
    gtk_main();

    return 0;
}

/*
 * ShowLabels
 *
 * Is the "show Labels" toolbar button pressed in to
 * indicate that the user wants to display the
 * atom labels?
 */
int ShowLabels ()
{

    return (GTK_TOGGLE_BUTTON (tool_labels)->active);
}

/*
 * ShowRGB
 *
 * Is the "show RGB" button on the toolbar pressed in?
 * This indicates whether the atoms should be displayed
 * in single color or multicolor mode.
 */
int ShowRGB ()
{

    return (GTK_TOGGLE_BUTTON (tool_rgb)->active);
}



/*
 * ShowLines
 *
 * Indicates whether the lines should be drawn between
 * the atoms to show the bonds.
 */
int ShowLines ()
{

    return (GTK_TOGGLE_BUTTON (tool_lines)->active);
```

```
}

/*
 * FileOpen
 *
 * Open a pdb file for displaying
 */
void FileOpen (GtkWidget *widget, gpointer data)
{
    GetFilename ("Open Molecule", ReadMolecule);
}

/*
 * CreateMainWindow
 *
 * Create the main window and the menu/toolbar associated with it
 */
static void CreateMainWindow ()
{
    GtkWidget *widget;
    GtkWidget *vbox_main;
    GtkWidget *menubar;
    GtkWidget *menu;
    GtkWidget *menuitem;
    GtkWidget *menufont;
    GtkWidget *toolbar;
    GtkWidget *button;

    /* --- Create the top window and size it --- */
    win_main = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_widget_set_usize(win_main, 360, 260);
    gtk_window_set_title (GTK_WINDOW (win_main), "Menu test");
    gtk_container_border_width (GTK_CONTAINER (win_main), 0);

    /* --- Create accel table --- */
    accelerator_table = gtk_accelerator_table_new();
    gtk_window_add_accelerator_table(GTK_WINDOW(win_main), accelerator_table);

    /* --- Top-level window should listen for the destroy --- */
    gtk_signal_connect (GTK_OBJECT (win_main), "destroy",
            GTK_SIGNAL_FUNC(EndProgram), NULL);
```

```
    /* --- Create v-box for menu, toolbar --- */
    vbox_main = gtk_vbox_new(FALSE, 0);

    /* --- Put up v-box --- */
    gtk_container_add(GTK_CONTAINER(win_main), vbox_main);

    gtk_widget_show(vbox_main);
    gtk_widget_show(win_main);

    /* --- Menu bar --- */
    menubar = gtk_menu_bar_new();
    gtk_box_pack_start(GTK_BOX(vbox_main), menubar, FALSE, TRUE, 0);
    gtk_widget_show(menubar);

    /* ------------------- */
    /* --- File menu --- */
    /* ------------------- */

    menu = CreateBarSubMenu(menubar, "File");

    menuitem = CreateMenuItem(menu, "Open","O",
                    "Open an existing item",
                    GTK_SIGNAL_FUNC(FileOpen), "open");

    menuitem = CreateMenuItem(menu, NULL, NULL,
                    NULL, NULL, NULL);

    menuitem = CreateMenuItem(menu, "Quit", "",
                    "What's more descriptive than quit?",
                    GTK_SIGNAL_FUNC(EndProgram), "quit");

    /* --- Create the toolbar --- */
    CreateToolbar(vbox_main);

    widget = CreateDrawingArea();
    gtk_box_pack_start(GTK_BOX(vbox_main), widget, TRUE, TRUE, 0);
}


/*
 * CreateToolbar
 *
```

```
 * Create the toolbar with the specified toolbar
 * options.
 */
void CreateToolbar (GtkWidget *vbox_main)
{
    GtkWidget *widget;

    /* --- Create the toolbar and add it to the window --- */
    toolbar = gtk_toolbar_new (GTK_ORIENTATION_HORIZONTAL, GTK_TOOLBAR_ICONS);
    gtk_box_pack_start (GTK_BOX (vbox_main), toolbar, FALSE, TRUE, 0);
    gtk_widget_show (toolbar);

    /* --- Create "Open" button --- */
    gtk_toolbar_append_item (GTK_TOOLBAR (toolbar),
                             "Open Dialog","Open dialog","",
                             CreateWidgetFromXpm (vbox_main, (gchar **) xpm_open),
                             (GtkSignalFunc) FileOpen,
                             NULL);

    /* --- Little gap --- */
    gtk_toolbar_append_space (GTK_TOOLBAR (toolbar));


    tool_lines = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar),
                             GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                             NULL,
                             "Lines","Lines","Lines",
                             CreateWidgetFromXpm (vbox_main, (gchar **) xpm_lines),
                             (GtkSignalFunc) MoleculeRepaint,
                             NULL);

    tool_rgb = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar),
                             GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                             NULL,
                             "Color","Color","Color",
                             CreateWidgetFromXpm (vbox_main, (gchar **) xpm_rgb),
                             (GtkSignalFunc) MoleculeRepaint,
                             NULL);

    tool_labels = gtk_toolbar_append_element (GTK_TOOLBAR (toolbar),
                             GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
                             NULL,
```

```
                "Labels","Labels","Labels",
                CreateWidgetFromXpm(vbox_main,(gchar **)xpm_labels),
                (GtkSignalFunc)MoleculeRepaint,
                NULL);

}
```

12.4.8.4 matrix3d.c

matrix3d.c 程序具有进行矩阵乘法所需要的函数。为了更好地理解 3 维数学,请查阅有关的专著(线性代数应该解释矩阵乘法;许多有关 3D 图形方面的书也应该包含这方面的内容)。为了完整起见,包含了下面的程序。

```
/*
 * File: matrix3d.c
 * Auth: Eric Harlow
 *
 * Converted from a Java Matrix 3d class file.
 */

#include "atom.h"
#include "matrix3d.h"
#include <math.h>


static double pi = 3.14159265;

/*
 * NewMatrix3D
 *
 * Create a new matrix
 */
typMatrix3D *NewMatrix3D()
{
    typMatrix3D *mat;

    mat = (typMatrix3D *)g_malloc(sizeof(typMatrix3D));

    unit(mat);

    return(mat);
}
```

```
/*
 * scale
 *
 * Scale the object
 */
void scale (typMatrix3D * mat, double f)
{
    mat->xx * = f;
    mat->xy * = f;
    mat->xz * = f;
    mat->xo * = f;
    mat->yx * = f;
    mat->yy * = f;
    mat->yz * = f;
    mat->yo * = f;
    mat->zx * = f;
    mat->zy * = f;
    mat->zz * = f;
    mat->zo * = f;
}


/*
 * scale3
 *
 * Scale each direction by a different factor
 */
void scale3 (typMatrix3D * mat, double xf, double yf, double zf)
{
    mat->xx * = xf;
    mat->xy * = xf;
    mat->xz * = xf;
    mat->xo * = xf;
    mat->yx * = yf;
    mat->yy * = yf;
    mat->yz * = yf;
    mat->yo * = yf;
    mat->zx * = zf;
    mat->zy * = zf;
    mat->zz * = zf;
    mat->zo * = zf;
}
```

```
/*
 * translate
 *
 * Translate the point represented by the matrix by (x, y, z)
 */
void translate (typMatrix3D  *mat, double x, double y, double z)
{
    mat->xo += x;
    mat->yo += y;
    mat->zo += z;
}

/*
 * yrot
 *
 * Add in a y rotation of angle (theta) to the matrix.
 */
void yrot (typMatrix3D *mat, double theta)
{
    double ct;
    double st;
    double Nxx;
    double Nxy;
    double Nxz;
    double Nxo;
    double Nzx;
    double Nzy;
    double Nzz;
    double Nzo;

    theta *= (pi / 180);
    ct = cos (theta);
    st = sin (theta);

    Nxx = (double) (mat->xx * ct + mat->zx * st);
    Nxy = (double) (mat->xy * ct + mat->zy * st);
    Nxz = (double) (mat->xz * ct + mat->zz * st);
    Nxo = (double) (mat->xo * ct + mat->zo * st);

    Nzx = (double) (mat->zx * ct - mat->xx * st);
```

```
    Nzy = (double) (mat->zy * ct - mat->xy * st);
    Nzz = (double) (mat->zz * ct - mat->xz * st);
    Nzo = (double) (mat->zo * ct - mat->xo * st);

    mat->xo = Nxo;
    mat->xx = Nxx;
    mat->xy = Nxy;
    mat->xz = Nxz;
    mat->zo = Nzo;
    mat->zx = Nzx;
    mat->zy = Nzy;
    mat->zz = Nzz;
}



/*
 * xrot
 *
 * Add in an x rotation of angle (theta) to the matrix.
 */
void xrot (typMatrix3D *mat, double theta)
{
    double ct;
    double st;
    double Nyx;
    double Nyy;
    double Nyz;
    double Nyo;
    double Nzx;
    double Nzy;
    double Nzz;
    double Nzo;

    theta *= (pi / 180);
    ct = cos (theta);
    st = sin (theta);

    Nyx = (double) (mat->yx * ct + mat->zx * st);
    Nyy = (double) (mat->yy * ct + mat->zy * st);
    Nyx = (double) (mat->yz * ct + mat->zz * st);
    Nyo = (double) (mat->yo * ct + mat->zo * st);
```

```
    Nzx = (double) (mat->zx * ct - mat->yx * st);
    Nzy = (double) (mat->zy * ct - mat->yy * st);
    Nzz = (double) (mat->zz * ct - mat->yz * st);
    Nzo = (double) (mat->zo * ct - mat->yo * st);

    mat->yo = Nyo;
    mat->yx = Nyx;
    mat->yy = Nyy;
    mat->yz = Nyz;
    mat->zo = Nzo;
    mat->zx = Nzx;
    mat->zy = Nzy;
    mat->zz = Nzz;
}


/*
 * zrot
 *
 * Add in a z rotation of angle (theta) to the matrix.
 */
void zrot (typMatrix3D *mat, double theta)
{
    double ct;
    double st;
    double Nyx;
    double Nyy;
    double Nyz;
    double Nyo;
    double Nxx;
    double Nxy;
    double Nxz;
    double Nxo;

    theta *= (pi / 180);
    ct = cos(theta);
    st = sin(theta);

    Nyx = (double) (mat->yx * ct + mat->xx * st);
    Nyy = (double) (mat->yy * ct + mat->xy * st);
    Nyz = (double) (mat->yz * ct + mat->xz * st);
```

```
    Nyo = (double) (mat->yo * ct + mat->xo * st);

    Nxx = (double) (mat->xx * ct - mat->yx * st);
    Nxy = (double) (mat->xy * ct - mat->yy * st);
    Nxz = (double) (mat->xz * ct - mat->yz * st);
    Nxo = (double) (mat->xo * ct - mat->yo * st);

    mat->yo = Nyo;
    mat->yx = Nyx;
    mat->yy = Nyy;
    mat->yz = Nyz;
    mat->xo = Nxo;
    mat->xx = Nxx;
    mat->xy = Nxy;
    mat->xz = Nxz;
}

/*
 * mult
 *
 * Multiply the first matrix by the second matrix.
 * The first matrix has the new value.
 */
void mult (typMatrix3D *mat, typMatrix3D *rhs)
{
    double lxx = mat->xx * rhs->xx +
                 mat->yx * rhs->xy +
                 mat->zx * rhs->xz;
    double lxy = mat->xy * rhs->xx +
                 mat->yy * rhs->xy +
                 mat->zy * rhs->xz;
    double lxz = mat->xz * rhs->xx +
                 mat->yz * rhs->xy +
                 mat->zz * rhs->xz;
    double lxo = mat->xo * rhs->xx +
                 mat->yo * rhs->xy +
                 mat->zo * rhs->xz + rhs->xo;

    double lyx = mat->xx * rhs->yx +
                 mat->yx * rhs->yy +
                 mat->zx * rhs->yz;
    double lyy = mat->xy * rhs->yx +
```

```
                mat->yy * rhs->yy +
                mat->zy * rhs->yz;
    double lyz = mat->xz * rhs->yx +
                mat->yz * rhs->yy +
                mat->zz * rhs->yz;
    double lyo = mat->xo * rhs->yx +
                mat->yo * rhs->yy +
                mat->zo * rhs->yz + rhs->yo;

    double lzx = mat->xx * rhs->zx +
                mat->yx * rhs->zy +
                mat->zx * rhs->zz;
    double lzy = mat->xy * rhs->zx +
                mat->yy * rhs->zy +
                mat->zy * rhs->zz;
    double lzz = mat->xz * rhs->zx +
                mat->yz * rhs->zy +
                mat->zz * rhs->zz;
    double lzo = mat->xo * rhs->zx +
                mat->yo * rhs->zy +
                mat->zo * rhs->zz + rhs->zo;

    mat->xx = lxx;
    mat->xy = lxy;
    mat->xz = lxz;
    mat->xo = lxo;

    mat->yx = lyx;
    mat->yy = lyy;
    mat->yz = lyz;
    mat->yo = lyo;

    mat->zx = lzx;
    mat->zy = lzy;
    mat->zz = lzz;
    mat->zo = lzo;
}


/*
 * unit
 *
```

```
 * Make the matrix a unit matrix
 */
void unit (typMatrix3D *mat)
{
    mat->xo = 0;
    mat->xx = 1;
    mat->xy = 0;
    mat->xz = 0;
    mat->yo = 0;
    mat->yx = 0;
    mat->yy = 1;
    mat->yz = 0;
    mat->zo = 0;
    mat->zx = 0;
    mat->zy = 0;
    mat->zz = 1;
}

/*
 * Transform
 *
 * Translate the atom's coordinates into the tranlated
 * coordinates.
 */
void Transform (typMatrix3D *mat, typAtom *atom)
{
    double lxx = mat->xx, lxy = mat->xy, lxz = mat->xz, lxo = mat->xo;
    double lyx = mat->yx, lyy = mat->yy, lyz = mat->yz, lyo = mat->yo;
    double lzx = mat->zx, lzy = mat->zy, lzz = mat->zz, lzo = mat->zo;

    double x = atom->x;
    double y = atom->y;
    double z = atom->z;

    atom->tx = (x * lxx + y * lxy + z * lxz + lxo);
    atom->ty = (x * lyx + y * lyy + z * lyz + lyo);
    atom->tz = (x * lzx + y * lzy + z * lzz + lzo);
}
```

12.4.9　程序的其余部分

程序的其余部分来自前面各章。程序 misc.c 来自第 5 章，用来建立菜单和工具条。程序

filesel.c 来自第 6 章。只需要对这些模块进行少量的改动就可以用在这里。对 filesel.c 所进行的修改使它更为通用。

12.5 小结

分子观察程序是用 GDK 建立的更为复杂的应用程序的例子。它说明在绘图区构件中如何用 GDK 实现人机通信功能,观察者利用鼠标事件旋转对象。本章说明如何通过低层事件对使用鼠标操纵分子旋转的用户作出响应。

第 13 章　可移动影像和动画制作

绘图工具集(GDK)不仅可以绘制线条和框,对它作一定的改进,它还能实现供游戏用的动画制作任务。从编写游戏的角度看,GDK 并不属于高性能的库。用它编写 Doom 和 Quake 一类的游戏可能并不理想。但是,编写性能要求不高的游戏仍然可以使用 GDK。

13.1　动画

计算机游戏中的动画是由在屏幕上快速移动的影像实现的。这一类影像可以是太空船,人或者任何其他能代表游戏者和计算机的任何东西。可移动影像有时称为 sprite。

只要屏幕上的影像不出现闪烁,动画的效果就好。防止闪烁的最好办法是利用双缓冲区技术(回忆第 10 章时钟的例子)。在本章中,活动序列(animated sequence)使用了像元图。这是因为这种格式比较熟悉,而且在背景的透明区上你能够容易地绘制它们。

实现活动序列包括 3 个步骤:第 1 步是清除背景和绘制背景的景色。第 2 步是绘制活动的影像。第 3 步是将绘制的情节传送到窗口,使它得到显示。

13.1.1　利用可移动影像

为了制作动画,首先需要将可移动影像定义为 .xpm 影像,从它可以得到可显示的像元图和一个掩模。掩模定义你需要绘制的区域。虽然影像是矩形的,但你并不需要在背景上绘制整个矩形。好的做法是屏蔽非影像区,你就不必去绘制没有影像数据的部分。

一个走动的人可以用 3 个影像产生:双腿并立、左腿向前和右腿向前。可以在左腿向前转向右腿向前之间以及右腿向前转向左腿向前之间分别插入双腿并立,以此实现人的走动。人的一个手中拿一本书来表示手臂的摆动。可以使用的影像定义如下:

```
static char *xpm_standright[] = {
"24 24 4 1",
"  c None",
"X c #FFFFFF",
". c #C00000",
"b c #C000FF",
"          ......          ",
"        .XXXXXX.          ",
"       .XXXXX.XX.         ",
"       .XXXXXXXX.         ",
"        .XXXXXX.          ",
"          .XX.            ",
"         .XXXX.           ",
```

```
"          .X.XX.          ",
"          .X.XX.          ",
"          .X.XX.          ",
"          .X.XX.          ",
"          .X.XX.          ",
"          .X.XX.          ",
"           .b..           ",
"           .bb.           ",
"           .bb.           ",
"           .bb.           ",
"           .bb.           ",
"           .bb.           ",
"           .bb.           ",
"           .bb...         ",
"           .XXXXX.        ",
"           .XXXXX.        ",
"            .....         ",
};


static char *xpm_walkright1[] = {
"24 24 4 1",
"  c None",
"X c #FFFFFF",
". c #000000",
"b c #0000FF",
"          ......          ",
"         .XXXXXX.         ",
"        .XXXXX.XX.        ",
"        .XXXXXXXX.        ",
"         .XXXXXX.         ",
"           .XX.           ",
"          .XXXX.          ",
"          .XXXX.          ",
"         .X.XX.X.         ",
"         .X.XX.X..        ",
"        .X..XX..XX.       ",
"        .X..XX.XXXX.      ",
"         .  .XX..XX.      ",
"           .bb.  ..       ",
"           ..b.           ",
"          .b..b.          ",
```

```
"          .b..b.          ",
"         .b.  .b.         ",
"         .b.  .b.         ",
"        .b.    .b.        ",
"        .b.    .b...      ",
"        .XXX.  .XXX.      ",
"        .XXX.  .XXX.      ",
"         ...    ...       ",
};


static char * xpm_walkright2[] = {
"24 24 4 1",
"  c None",
"X c #FFFFFF",
". c #000000",
"b c #0000FF",
"          ......         ",
"         .XXXXXX.        ",
"        .XXXXX.XX.       ",
"        .XXXXXXXX.       ",
"         .XXXXXX.        ",
"           .XX.          ",
"         .XXXX.          ",
"         .XXXX.          ",
"        .X.XX.X.         ",
"       ..X.XX.X.         ",
"      .XX.  XX..X.       ",
"     .XXXX.XX..X.        ",
"      .XX..XX.  .        ",
"       ..  .bb.          ",
"           .b..          ",
"         .b..b.          ",
"         .b..b.          ",
"        .b.  .b.         ",
"         .b.  .b.        ",
"        .b.    .b.       ",
"        .b..   .b..      ",
"        .XXX.  .XXX.     ",
"        .XXX.  .XXX.     ",
"         ...    ...      ",
};
```

你可以将影像保存在 sprite 数据结构中。sprite 数据结构的定义如下:

```
typedef struct {
    char **xpm_data;
    GdkPixmap *pixmap;
    GdkBitmap *mask;
} typSprite;
```

其中 xpm_data 是影像的 .xpm 像元图数据,而调用 gdk_pixmap_create_from_xmp_d 函数则产生像元图和掩模。走动的人有一个 typSprite 数组,前面的一些影像放在数组中;并将结构的像元图和掩模初始化为 NULL。

```
/*
 * Man walking
 */
typSprite walk[] =  {
    { xpm_standright, NULL, NULL },
    { xpm_walkright1, NULL, NULL },
    { xpm_standright, NULL, NULL },
    { xpm_walkright2, NULL, NULL },
    { NULL, NULL, NULL }
};
```

sprite 数组定义一系列影像以及这些影像显示的顺序。typActor 结构定义影像在屏幕上的位置,在序列中显示什么影像以及影像的大小,假定所有在序列中的可移动影像的尺寸相同。

```
typedef struct {

    int seq;
    int x;
    int y;
    int height;
    int width;
    int nSequences;
    typSprite *sprites;

} typActor;
```

13.1.2 装入影像

函数 LoadPixmap 将 typActor 和 typSprite 联系在一起。所以,演员(actor)能够与显示在屏幕上的可移动影像和它的位置保持一致。可以使用窗口,actor 和将要显示 actor 的可移动影像作为参数装入像元图。其中,可移动影像数组 walk 与 actor 联系在一起,并进行初始化。

```
LoadPixmaps (window, &man, walk);
```

LoadPixmaps 函数将每个 .xpm 数据影像转换为像元图和掩模数据。将可移动影像数组保存在 typActor 结构中，用序列中的可移动影像的数目和影像大小对 typActor 进行初始化。

```
void LoadPixmaps (GtkWidget *widget,
                  typActor *actor,
                  typSprite *sprites)
{

    int i = 0;

    /* --- Get every sequence --- */
    while (sprites[i].xpm_data) {

        /* --- Convert xpm data to pixmap & mask --- */
        sprites[i].pixmap = gdk_pixmap_create_from_xpm_d (
                                widget->window,
                                &sprites[i].mask,
                                NULL,
                                sprites[i].xpm_data);
        i++;
    }

    /* --- Get sprite height and width --- */
    GetWidthHeight (sprites[0].xpm_data, &actor->width,
                    &actor->height);

    /* --- Initialize sprite information --- */
    actor->seq = 0;
    actor->nSequences = i;
    actor->sprites = sprites;
}
```

13.1.3　显示影像

在所有演员已经和可移动影像结合在一起后，动画已经可以开始了。和其他双缓冲区的例子一样，repaint 子程序在背景上进行绘制。当绘制工作结束时，背景影像就被复制到绘图区。

```
gint Repaint (gpointer data)

{
    GtkWidget *     drawing_area = (GtkWidget *) data;
```

```
GdkRectangle  update_rect;
static offset = 0;
int nTop;

/* --  clear pixmap (background image) --- */
gdk_draw_rectangle (pixmap,
          drawing_area->style->black_gc,
          TRUE,
          0, 0,
          drawing_area->allocation.width,
          drawing_area->allocation.height);

/*  -- Draw road --- */
gdk_draw_rectangle (pixmap,
          drawing_area->style->white_gc,
          TRUE,
          50, 0,
          100,
          drawing_area->allocation.height);

/*
 * Draw lines on the road
 */

/* --- Figure out where first line should be --- */
offset++;
if ((offset - GAP_LEN) >= 0) {
    offset -= (LINE_LEN + GAP_LEN);
}

/* --- Draw the lines all the way down the road --- */
nTop = offset;
do {
    gdk_draw_rectangle (pixmap,
          drawing_area->style->black_gc,
          TRUE,
          100, nTop, 5, LINE_LEN);
    nTop += LINE_LEN + GAP_LEN;
} while (nTop < drawing_area->allocation.height);

/* --- Draw each of these images --- */
SequenceSprite (drawing_area, &bike, 3);
```

```
    SequenceSprite (drawing_area, &man, 2);
    SequenceSprite (drawing_area, &woman, 2);
    SequenceSprite (drawing_area, &police, 0);
    SequenceSprite (drawing_area, &ball1, 2);
    SequenceSprite (drawing_area, &car1, 3);

    /* --- The whole screen needs to be updated --- */
    update_rect.x = 0;
    update_rect.y = 0;
    update_rect.width = drawing_area->allocation.width;
    update_rect.height = drawing_area->allocation.height;

    /* --- So update it --- */
    gtk_widget_draw (drawing_area, &update_rect);
}
```

函数 SequenceSprite 在背景上绘制每一个活动的影像，并利用 gdk_draw_pixmap 绘制影像。通常在背景上绘制影像时将影像绘制在一个矩形区域。(见图 13.1)。

利用 gdk_pixmap_create_from_xpm_d 函数返回的掩模，可以不必绘制非影像数据，你可利用 .xpm 数据中的颜色建立掩模。颜色 none 是建立掩模时唯一不使用的颜色，因而在掩模中成为透明。函数 gdk_gc_set_clip_mask 设置 gdk_draw_pixmap 函数使用的掩模。利用 gdk_gc_set_clip_mask 函数，掩模必须位于将要绘制影像的地方。当在页面上绘制影像时，利用这两个函数消除了围绕影像周围的方框(见图 13.2)。

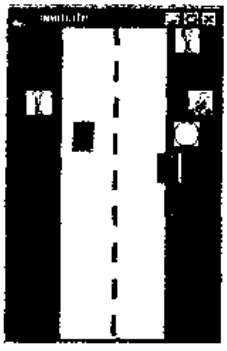


图 13.1　不透明的可移动影像



图 13.2　透明的可移动影像

函数 SequenceSprite 取演员数据，给它的 x 值增加一个距离(nDist)，并在屏幕上新的位置绘制它。在绘制时，可以使用掩模或者不使用掩模。用 bUseMask 标志表示是否在绘制可移动影像时使用掩模。不用掩模进行绘制将导致在影像周围出现一个大的矩形(参看图 13.1)。当然，如果在绘制影像时使用剪下的掩模(clipping mask)，在绘制影像后不再需要它时，必须将它设置为 NULL。下面是完整的 SequenceSprite 函数。

```
SequenceSprite (GtkWidget * drawing_area, typActor * actor, int nDist)
{
    GdkGC *gc;

    /* --- Move the actor along --- */
    actor->x += nDist;

    /* --- Too far over, reset position --- */
    if (actor->x > drawing_area->allocation.width) {
        actor->x = 0;
    }

    /* --- Use the next image for the actor --- */
    actor->seq++;

    /* --- Use 0 if at the end --- */
    if (actor->seq >= actor->nSequences) {
        actor->seq = 0;
    }

    /* --- Get foreground color --- */
    gc = drawing_area->style->fg_gc[GTK_STATE_NORMAL];

    if (bUseMask) {

        /* --- Set the clipping of the sprites --- */
        gdk_gc_set_clip_mask (gc, actor->sprites[actor->seq].mask);

        /* --- Set the origin of the clipping --- */
        gdk_gc_set_clip_origin (gc, actor->x, actor->y);
    }

    /* --- Copy pixmap to the window, properly clipped --- */
    gdk_draw_pixmap (pixmap,
          drawing_area->style->fg_gc[GTK_STATE_NORMAL],
          actor->sprites[actor->seq].pixmap,
          0, 0,
          actor->x, actor->y,
          actor->width, actor->height);

    if (bUseMask) {
```

```
        /* --- Clear the clipping mask --- */
        gdk_gc_set_clip_mask(gc, NULL);
    }
}
```

13.1.4　整个程序

整个动画演示程序如下。它使用若干活动的序列，并将它们移过窗口。大多数可移动影像是一系列像元图，用来使影像呈现好象在移动一样。在影像中，走动的人看上去像一步步在向前走动。骑自行车的人好象踩着自行车前进。球的可移动影像表示为在对象内部有透明的区域。当球移过背景和其他可移动影像时，可以通过球的洞看到其他对象。

```
/*
 * Auth: Eric Harlow
 * Linux application development
 *
 * Demo of sprites.
 */

#include <gtk/gtk.h>
#include <time.h>

#include "man.h"
#include "woman.h"
#include "ball.h"
#include "driver.h"
#include "police.h"
#include "bike.h"


#define LINE_LEN 20
#define GAP_LEN 15

/*
 * Our sprite data structure
 */
typedef struct {

    char **xpm_data;
    GdkPixmap *pixmap;
    GdkBitmap *mask;
```

```
} typSprite;

/*
 * Here's the actor. Actor consists of one or more
 * sprites.
 */
typedef struct {

    int seq;
    int x;
    int y;
    int height;
    int width;
    int nSequences;
    typSprite * sprites;

} typActor;

/*
 * Man walking
 */
typSprite walk[] = {
    { xpm_standright, NULL, NULL },
    { xpm_walkright1, NULL, NULL },
    { xpm_standright, NULL, NULL },
    { xpm_walkright2, NULL, NULL },
    { NULL, NULL, NULL }
};

/*
 * Man on a bike
 */
typSprite sprite_bike[] = {
    { xpm_bike1, NULL, NULL },
    { xpm_bike2, NULL, NULL },
    { xpm_bike3, NULL, NULL },
    { NULL, NULL, NULL }
};


/*
 * Woman walking
```

```
 */
typSprite sprite_woman[] = {
    { xpm_womanr, NULL, NULL },
    { xpm_womanwalkr1, NULL, NULL },
    { xpm_womanr, NULL, NULL },
    { xpm_womanwalkr2, NULL, NULL },
    { NULL, NULL, NULL }
};


/*
 * Police car
 */
typSprite sprite_police[] = {
    { xpm_police1, NULL, NULL },
    { xpm_police2, NULL, NULL },
    { NULL, NULL, NULL }
};


/*
 * Partially transparent ball.
 */
typSprite sprite_ball[] = {
    { xpm_ball1, NULL, NULL },
    { NULL, NULL, NULL }
};

/*
 * Car
 */
typSprite sprite_car[] = {
    { xpm_car1, NULL, NULL },
    { xpm_car1, NULL, NULL },
    { NULL, NULL, NULL }
};


/*
 * Here are the stars of the production
 */
typActor man;
```

```
typActor bike;
typActor woman;
typActor police;
typActor ball1;
typActor car1;

/* --- Backing pixmap for drawing area --- */
static GdkPixmap *pixmap = NULL;

/* --- Flag to use mask --- */
static int bUseMask = TRUE;

/*
 * Prototypes.
 */
void GetWidthHeight (gchar **xpm, int *width, int *height);


/*
 * LoadPixmaps
 *
 * Load the actor's pixmaps, get the sprite information
 * from the xpm data and initialize the actor animation
 * information.
 */
void LoadPixmaps (GtkWidget *widget, typActor *actor, typSprite *sprites)
{

    int i = 0;

    /* --- Get every sequence --- */
    while (sprites[i].xpm_data) {

        /* --- Convert .xpm data to pixmap & mask --- */
        sprites[i].pixmap = gdk_pixmap_create_from_xpm_d (
                                widget->window,
                                &sprites[i].mask,
                                NULL,
                                sprites[i].xpm_data);
        i++;
    }
```

```
    /* --- Get sprite height and width --- */
    GetWidthHeight (sprites[0].xpm_data, &actor->width, &actor->height);

    /* --- Initialize sprite information --- */
    actor->seq = 0;
    actor->nSequences = i;
    actor->sprites = sprites;
}


/*
 * SequenceSprite
 *
 * Move to the next sprite in the sequence and
 * draw it with the mask
 */
SequenceSprite (GtkWidget *drawing_area, typActor *actor, int nDist)
{
    GdkGC *gc;

    actor->x += nDist;
    if (actor->x > drawing_area->allocation.width) {
        actor->x = 0;
    }

    /* --- Use the next image for the actor --- */
    actor->seq++;

    /* --- Use 0 if at the end --- */
    if (actor->seq >= actor->nSequences) {
        actor->seq = 0;
    }

    /* --- Get foreground color --- */
    gc = drawing_area->style->fg_gc[GTK_STATE_NORMAL];

    if (bUseMask) {

        /* --- Set the clipping of the sprites --- */
        gdk_gc_set_clip_mask (gc, actor->sprites[actor->seq].mask);

        /* --- Set the origin of the clipping --- */
```

```
        gdk_gc_set_clip_origin (gc, actor->x, actor->y);
    }

    /* --- Copy pixmap to the window, properly clipped --- */
    gdk_draw_pixmap (pixmap,
          drawing_area->style->fg_gc[GTK_STATE_NORMAL],
          actor->sprites[actor->seq].pixmap,
          0, 0,
          actor->x, actor->y,
          actor->width, actor->height);

    if (bUseMask) {

        /* --- Clear the clipping mask --- */
        gdk_gc_set_clip_mask (gc, NULL);
    }
}


/*
 * Repaint
 *
 * data - widget to repaint
 */
gint Repaint (gpointer data)
{
    GtkWidget *     drawing_area = (GtkWidget *) data;
    GdkRectangle  update_rect;
    static offset = 0;
    int nTop;

    /* --- clear pixmap (background image) --- */
    gdk_draw_rectangle (pixmap,
              drawing_area->style->black_gc,
              TRUE,
              0, 0,
              drawing_area->allocation.width,
              drawing_area->allocation.height);

    /* --- Draw road --- */
    gdk_draw_rectangle (pixmap,
              drawing_area->style->white_gc,
```

```
                TRUE,
                50, 0,
                100,
                drawing_area->allocation.height);

    /*
     * Draw lines on the road
     */

    /* --- Figure out where first line should be. --- */
    offset++;
    if ((offset - GAP_LEN) >= 0) {
        offset -= (LINE_LEN + GAP_LEN);
    }

    /* --- Draw the lines all the way down the road --- */
    nTop = offset;
    do {
        gdk_draw_rectangle (pixmap,
                drawing_area->style->black_gc,
                TRUE,
                100, nTop, 5, LINE_LEN);
        nTop += LINE_LEN + GAP_LEN;
    } while (nTop < drawing_area->allocation.height);

    /* --- Draw each of these images --- */
    SequenceSprite (drawing_area, &bike, 3);
    SequenceSprite (drawing_area, &man, 2);
    SequenceSprite (drawing_area, &woman, 2);
    SequenceSprite (drawing_area, &police, 0);
    SequenceSprite (drawing_area, &ball1, 2);
    SequenceSprite (drawing_area, &car1, 3);

    /* --- The whole screen needs to be updated --- */
    update_rect.x = 0;
    update_rect.y = 0;
    update_rect.width = drawing_area->allocation.width;
    update_rect.height = drawing_area->allocation.height;

    /* --- So update it --- */
    gtk_widget_draw (drawing_area, &update_rect);
}
```

```
/*
 * configure_event
 *
 * Create a new backing pixmap of the appropriate size
 * Of course, this is called whenever the window is
 * resized.  We have to free up things we allocated.
 */
static gint configure_event (GtkWidget *widget, GdkEventConfigure *event)
{
    /* --- Free background if we created it --- */
    if (pixmap) {
        gdk_pixmap_unref (pixmap);
    }

    /* --- Create a new pixmap with new size --- */
    pixmap = gdk_pixmap_new (widget->window,
                widget->allocation.width,
                widget->allocation.height,
                -1);

    return TRUE;
}

/*
 * expose_event
 *
 * When the window is exposed to the viewer or
 * the gdk_widget_draw routine is called, this
 * routine is called.  Copies the background pixmap
 * to the window.
 */
gint expose_event (GtkWidget *widget, GdkEventExpose *event)
{

    /* --- Copy pixmap to the window --- */
    gdk_draw_pixmap (widget->window,
          widget->style->fg_gc[GTK_WIDGET_STATE (widget)],
          pixmap,
          event->area.x, event->area.y,
          event->area.x, event->area.y,
          event->area.width, event->area.height);
```

```
    return FALSE;
}


/*
 * quit
 *
 * Get out of the application
 */
void quit ()
{
    gtk_exit (0);
}


/*
 * GetWidgthHeight
 *
 * Get the widget and height for the xpm data
 *
 */
void GetWidthHeight (gchar **xpm, int *width, int *height)
{

        sscanf (xpm[0], "%d %d", width, height);
}


/*
 * main
 *
 * Program begins here.
 */
int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *drawing_area;
    GtkWidget *vbox;

    /* --- Initialize GTK+ --- */
    gtk_init (&argc, &argv);
```

```
if (argc > 1) {
    bUseMask = FALSE;
}

/* --- Create the top-level window --- */
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

vbox = gtk_vbox_new (FALSE, 0);
gtk_container_add (GTK_CONTAINER (window), vbox);
gtk_widget_show (vbox);

gtk_signal_connect (GTK_OBJECT (window), "destroy",
            GTK_SIGNAL_FUNC (quit), NULL);

/* --- Create the drawing area --- */
drawing_area = gtk_drawing_area_new ();
gtk_drawing_area_size (GTK_DRAWING_AREA (drawing_area), 200, 300);
gtk_box_pack_start (GTK_BOX (vbox), drawing_area, TRUE, TRUE, 0);

gtk_widget_show (drawing_area);

/* --- Signals used to handle backing pixmap --- */
gtk_signal_connect (GTK_OBJECT (drawing_area), "expose_event",
            (GtkSignalFunc) expose_event, NULL);
gtk_signal_connect (GTK_OBJECT(drawing_area),"configure_event",
            (GtkSignalFunc) configure_event, NULL);

/* --- Show the window --- */
gtk_widget_show (window);

/* --- Repaint every so often --- */
gtk_timeout_add (100, Repaint, drawing_area);

/* --- Load all these sprites --- */
LoadPixmaps (window, &man, walk);
LoadPixmaps (window, &bike, sprite_bike);
LoadPixmaps (window, &woman, sprite_woman);
LoadPixmaps (window, &police, sprite_police);
LoadPixmaps (window, &ball1, sprite_ball);
LoadPixmaps (window, &car1, sprite_car);
```

```
    /* --- Position them --- */
    bike.x = 30;
    bike.y = 60;

    man.x = 50;
    man.y = 60;

    man.x = 60;
    man.y = 60;

    police.x = 60;
    police.y = 90;

    ball1.x = 0;
    ball1.y = 90;

    car1.x = 0;
    car1.y = 120;

    /* --- Call gtk-main loop --- */
    gtk_main ();

    return 0;
}
```

13.2　电视游戏

能够用 GDK 实现的远不止简单的动画。你可以利用从前一节中掌握的知识建立游戏模型的基础。这里使用的游戏模型模仿了 80 年代电视娱乐中心的游戏《防御者》("Defender")。年轻人可能回忆不起来了,《防御者》是一种游戏,其中英雄在地球上空飞行保护他的人民不受入侵的外星人的蹂躏。外星人试图把人从地球表面带走,并将他们带到屏幕上方的区域。在这一区域中,外星人发生突变。突变体将消灭带到屏幕上方的人,并变为具有更大破坏能力的外星人。

我不得不从记忆中重新创造这一游戏(已经无法从当地的娱乐中心找到这一古老的电视游戏)。由于空间的限制,我也不得不取消游戏原有的一些特征。没有进行记分,原始游戏中的一些外星人也不见了。这里的目的只是演示 GDK 在这个可扩充的简短游戏中的能力。对游戏的扩充留给读者作为练习。

13.2.1　用 GTK+/GDK 实现的游戏

对这个程序来说,利用有关可移动影像和动画制作的知识编写游戏的框架。在《防御者》

游戏中使用的动画类似你在前面建立的动画。但是,现在游戏者控制一种可移动影像的行为,而计算机则控制另一种可移动影像的行为。而且在所有可移动影像之间可以进行对话。也许你会说:你能够利用 GTK + 和 GDK 产生比这个简化的《防御者》更有魅力的游戏。

实际的电视游戏要复杂得多。但是,这个例子说明 GDK + 能够用于电视游戏,而且对 CPU 的影响不大,特别是对比较新的硬件。在奔腾 133 上,这个游戏运行得很好。而且 CPU 负荷也很轻。当然在 386 或者 486SX-16 上它不会运行得很好。

13.2.2 建立游戏需要考虑的因素

在建立游戏时,有许多因素需要加以考虑:游戏的输入、图形、人工智能、特殊效果、冲突、级别的提升和记分等只是你在编写这类游戏需要加以考虑的一部分因素。例如,《防御者》游戏中没有实现记分,也没有提升游戏的级别。在实际游戏中,循序进入比较难的级别是必要的。缺乏变化和挑战性,游戏本身就会很快被淘汰。

13.2.2.1 输入

键盘是每个人必须具有的输入设备,所以这是游戏者在游戏中使用的设备。游戏具有移动(左、右、上和下)和向外星人开火的能力(见图 13.3)。移动可以用箭头键,开火可以用空格键,但它并不那么简短。写游戏不同于写应用程序。写游戏时,你需要知道经常按哪些键。例如,游戏者可能按下方向键,同时敲击空格键向外星人开火以击落外星人。在缺省情况下,用 GTK + 做不到这一点。因为最后按下的键是送给应用程序的键。如果用户按住一个键,就重复向应用程序发送这一个键。如果在按住一个键时再按其他键,其他键将被忽略。



图 13.3 着陆者在抓人——其中一个着陆者抓住和按倒了一个人

13.2.2.2 跟踪击键

可以利用"key _ press _ event"信号和"key _ release _ event"信号消除按键的问题。当按下一个键时,产生"key _ press _ event"信号;当释放一个键时,产生"key _ release _ event"信号。然而,在缺省的情况下,不发送"key _ release _ event"信号。所以只设置一个回呼函数并不能正常工作。为了确保得到事件信号,必须使用 GDK _ KEY _ RELEASE _ MASK 参数调用 gtk

_ widget _ set _ events 函数。这一步保证向窗口发送“key _ release _ event”信号。GTK + 通常屏蔽“不重要”的信号，当你实际需要这些信号时，必须明确告诉它发送这些信号。

13.2.2.3　重复键

这里不需要重复键。虽然按下多个键对应用程序的唯一效果是导致不必要的键的处理和奇怪的键击组合。函数 gdk _ key _ repeat _ disable 消除在应用程序中的重复键。在调用这个函数以后，当按下任何键时，它只得到一个“key _ press _ event”信号，即使用户按住了一个键情况也是如此。使用函数 gdk _ key _ repeat _ disable 所带来问题是它的效果是全程的——不仅对于游戏，而且对所有在运行中的应用程序都有影响。为了防止游戏影响其他应用程序，只允许在将应用作为重点(“focus _ in _ event”)时使用 gdk _ key _ repeat _ disable 函数；其中游戏是事件的重点。当重点转向其他应用时，“focus _ in _ event”的回呼函数将调用 gdk _ key _ repeat _ restore 函数恢复键的重复功能。例如，老板从用户旁边经过时，用户从玩弄游戏转向使用电子表格就自动恢复键的重复功能。

13.2.2.4　图形

游戏有几种图形要素：雷达屏幕显示游戏中的所有单元(unit)，主屏幕是发生大多数行动的地方，每个游戏片断的可移动影像以及一些特殊效果等。游戏中的每一个单元是从应用的 .xpm 文件装入的像元图。游戏采用双缓冲区技术以提供平滑的动画。对游戏的每一帧重新绘制整个屏幕。首先清除背景，然后绘制在屏幕上的各个单元，接着画雷达——显示所有的单元。当所有绘制工作完成时，将影像从背景转移到绘图区构件。

在主屏幕中的影像是用像元图绘制的。在雷达中的图形用彩色点进行绘制。每种颜色代表一种不同的单元。所以，过一会儿辨认不同单元是很方便的。即使外星人发射的导弹和爆炸都能显示在雷达屏幕中。

13.2.2.5　人工智能

在游戏中的每一个单元都有它自己的特性。在这个游戏中使用的人工智能(AI)是原始的，但它实现了使游戏具有挑战性的目标。特别是当增加更多的单元时，实际上你可能输掉这场游戏。(禁止使用对游戏者的冲突检测)。

在游戏中只有两个单元使用了 AI。每一个都试图完成特殊任务。着陆者沿水平方向移动搜索被捕的人，直到它发现某个人为止。当着陆者直接位于地面上人的上方时，它下降并试图将人带到屏幕上方，在那里它们(外星人)能够转变为突变体(见图 13.4)。突变体追捕游戏者。它们被设计成以某种随机方式移动，但始终朝向游戏者。

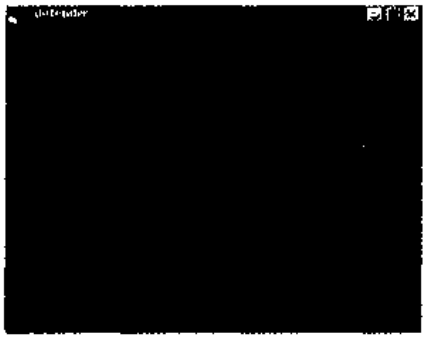


图 13.4 英雄射击一对突变体——其中一个被杀，另一个也将死亡

用于突变体的 AI 使突变体在 x 方向有 4 种移动方式——离开游戏者，停留在原来的位置，冲向游戏者以及以大步跳跃的方式冲向游戏者。百分之五十的时间突变体冲向游戏者。突变体还能以随机方式向后跳跃。但在战斗中突变体大部分时间冲向英雄。沿 y 方向的移动是类似的，但只发生在英雄出现在视距范围内时。否则沿 y 方向的移动是随机的。移动是由 AIModule 函数实现的。当英雄在距离内时，每一个单元都有开火的机会。当然，突变体开火的机会更多。

13.2.3 游戏内部的细节

游戏需要跟踪所有的特殊单元以及它们在地球上的位置。游戏的可移动影像和山是最重要的跟踪项目。在游戏中可移动影像是经常移动的。虽然山是不移动的，当英雄飞行时它好象在移动。它们的位置(相对于英雄的位置)是变化的。当英雄水平移动时，在屏幕上画的山是移动的。由于地球呈圆弧形，计算单元之间的距离和方向稍有一点复杂。为了简化，所有的单元都使用相同的数据结构，虽然它们并不都使用所有字段。

13.2.3.1 数据结构

两个基本的数据结构是 typSprite 数据结构和 typUnit。typSprite 包含像元图信息，其中包括可移动影像掩模、高度和宽度。typUnit 数据结构包含在游戏中有关所有单元的信息。游戏者、外星人、导弹以及爆炸都在 typUnit 中。TypUnit 保存单元的类型(着陆者、突变体、导弹和爆炸等)、单元的坐标、速度、生命周期以及向游戏者开火的机会。对特定单元的生命周期可能需要作一定解释。

大多数对象的生命周期是永恒的，或者至少持续到对象被摧毁；但是一些单元有预期的生命周期。例如，从外星人开火发出的导弹最终将"消失"，所以对它们给出生命周期。从导弹自外星人手中发出之时起开始递减，直到生命周期达到零自动消失。爆炸类似导弹，它也有预期的生命周期。但是，爆炸实际是由 8 个小的可移动影像组成，用它们向多个方向散开来模拟爆炸的效果。用链接表(GList *)跟踪除了游戏者/英雄以外的所有单元。因此，链接表包含外星人、导弹以及每个爆炸的所有 8 个碎片。

13.2.3.2　山

当外星人或人不出现在屏幕上时或者当它们被摧毁时，用山来衬托运动。在游戏中山作为参考点来模拟运动。用建立随机的山峰值(x,y)并用它们来建立形成山的山峰和山谷的点的列表。

13.2.3.3　计算

由于游戏是在从 0 到 X_RANGE 再回到 0 的虚拟的地球表面进行的，当计算距离和方向时需要一些附加的步骤。假定 X_RANGE 为 2000，如果游戏者的 x 值是 1999，而敌人的 x 值是 1，则它们之间的距离不是 1998，而是 2，因为地球从 1999 返回到 0。

13.2.3.4　数据类型

现在你已经了解程序需要做些什么，让我们看它如何实现。首先说明数据类型和图形，接着是相应的代码。

13.2.3.5　defender.h

```
/*
 * Directions the player can move.
 */
enum {
    MOVE_LEFT,
    MOVE_RIGHT,
    MOVE_UP,
    MOVE_DOWN
};


/*
 * The cast of characters in the game.
 */
enum {
HERO,      /* ---player--- */
PERSON,    /* ---people on the surface--- */
LANDER,    /* ---alien trying to pluck people--- */
MUTANT,    /* ---alien that took person to top and mutated--- */
MISSILE,   /* ---alien shoots at player--- */
LASER,     /* ---player shoots at alien--- */
EXPLOSION /* ---somebody goes kaboom--- */

};
```

```
/*
 * Data structure used by all game units.
 */
typedef struct unit {

   int bDestroy;          /* --- Scheduled to be destroyed --- */
   int direction;         /* --- Which direction is unit going? --- */
   int type;              /* --- Unit type --- */
   float pctFiring;       /* --- Chance of firing on player --- */
   float x;               /* --- Position --- */
   float y;               /* --- Position --- */
   float vx;              /* --- Velocity --- */
   float vy;              /* --- Velocity --- */
   int life;              /* --- Life left --- */
   struct unit *lockunit; /* --- Units locked to it --- */

} typUnit;



typedef struct {

    int x;
    int y;

} typPoint;


typedef struct {

    typPoint start;
    typPoint peak;
    typPoint end;

} typMountain;


/*
 * The sprites used to draw the units on the screen
 */
```

```
typedef struct {

    char **xpm_data;     /* ---Original xpm data--- */
    GdkPixmap *pixmap;   /* ---Pixmap--- */
    GdkBitmap *mask;     /* ---Pixmap mask--- */
    int height;          /* ---Height of sprite--- */
    int width;           /* ---Width of sprite--- */

} typSprite;
```

13.2.3.6　mutant.h

mutant.h 包含在游戏中的所有角色的图形。

```
/*
 * Graphics for all the characters in the
 * defender-like game.
 */

static char * xpm_missile[] = {
"2 2 2 1",
"       c None",
"X      c #ffffff",
"XX",
"XX",
};


static char * xpm_mutant[] = {
"16 12 4 1",
"       c None",
"b      c #8888ff",
"r      c #FF3366",
"G      c #00AA00",
"       bbbr      ",
"     rrbbbrrr    ",
"   GrrbbbrrrG    ",
"  GGr  rr  rGG   ",
"  GGr  rr  rGG   ",
"  GGrrrrrrrrGG   ",
"   GGrrrrrrGG    ",
"     GGGrrGGG    ",
```

```
"    GG rr GG    ",
"   GG  rr  GG   ",
"  GG   rr   GG  ",
" GG    rr    GG ",
};


static char * xpm_lander[] = {
"16 12 4 1",
"        c None",
"y       c #ffaa00",
"g       c #88FF88",
"G       c #009900",
"       yyyy     ",
"    GGyyyyGG    ",
"   GGggGGGG     ",
"  GGG ggg GGG   ",
"  GGG gg  GGG   ",
"  GGGgggggGGG   ",
"   GGGgggGGG    ",
"    GGGGGGG     ",
"    GG GG GG    ",
"   GG  GG  GG   ",
"  GG   GG   GG  ",
" GG    GG    GG ",
};

static char * xpm_ship1[] = {
"22 7 4 1",
"  c None",
"x c #777777",
"p c #ff66ff",
"o c #cccccc",
"    x                    ",
"   xxx                   ",
"  xxxxx                  ",
"xxxxxxxxxxxxxpp          ",
"xxxxxxxxxxxxxxxxxxxoo    ",
"ppppppxxxxxxxxxxxxxxxo   ",
" ppppppxxxo              "
};
```

```
static char * xpm_ship2 [] = {
"22 7 4 1",
"  c None",
"x c #777777",
"p c #ff66ff",
"o c #cccccc",
"                    x   ",
"                   xxx  ",
"                  xxxxx ",
"      ppxxxxxxxxxxxxxx",
" oxxxxxxxxxxxxxxxxxxx ",
"oooxxxxxxxxxxxxxpppppp ",
"            ooxxppppp ",
};


static char * xpm_man[] = {
"6 10 4 1",
"       c None",
"y      c #ffaa00",
"p      c #CC00CC",
"p      c #FF44FF",
"  pp  ",
"  yP  ",
" yPPP ",
" PPPP ",
" PPPP ",
" PPPP ",
"  pp  ",
"  pp  ",
"  pp  ",
"  pp  ",
};
```

13.2.3.7　animate.c

在 animate.c 中的程序不牵涉到任何游戏规则，它主要与建立窗口和处理事件有关。基本思想适用于任何游戏，现在它只用于仿制《防御者》的游戏中。

```
/*
 * Auth: Eric Harlow
```

```
 * File: animate.c
 *
 * Linux application development
 *
 * Defender game.
 */

#include <gtk/gtk.h>
#include <gdk/gdkkeysyms.h>
#include "defender.h"


/* --- Backing pixmap for drawing area --- */
GdkPixmap *pixmap = NULL;


/*
 * DisplaySprite
 *
 * Display the sprite on the drawing_area at the prescribed
 * coordinates.  Use a mask so that we do not draw the
 * invisible area - it should be transparent.
 *
 * drawing_area - where to draw the sprite
 * sprite - the sprite to draw
 * x, y - position to draw the sprite.
 */
void DisplaySprite (GtkWidget *drawing_area, typSprite *sprite, int x, int y)
{
    GdkGC *gc;
    int midx, midy;

    /* --- Get gc of the normal style --- */
    gc = drawing_area->style->fg_gc[GTK_STATE_NORMAL];

    /* --- Set up the clip mask and origin --- */
    gdk_gc_set_clip_mask (gc, sprite->mask);
    gdk_gc_set_clip_origin (gc, x, y);

    /* --- Copy pixmap to the window --- */
    gdk_draw_pixmap (pixmap,
          drawing_area->style->fg_gc[GTK_STATE_NORMAL],
```

```
            sprite->pixmap,
            0, 0, x, y,
            sprite->width, sprite->height);

    /* --- Clear the clip mask when done. --- */
    gdk_gc_set_clip_mask (gc, NULL);
}



/*
 * Repaint
 *
 * data - widget to repaint
 */
gint Repaint (gpointer data)
{
    GtkWidget *     drawing_area = (GtkWidget *) data;
    GdkRectangle  update_rect;
    static offset = 0;
    int rTop;

    /* --- Draw the game on the background. --- */
    DrawScreen (pixmap, drawing_area);

    /* --- Copy background to screen --- */
    update_rect.x = 0;
    update_rect.y = 0;
    update_rect.width = drawing_area->allocation.width;
    update_rect.height = drawing_area->allocation.height;

    /* --- Copy the background to the foreground --- */
    gtk_widget_draw (drawing_area, &update_rect);

    return TRUE;
}

/*
 * configure_event
 *
 * Create a new backing pixmap of the appropriate size.
 * Of course, this is called whenever the window is
```

```
 * resized.  We have to free up things we allocated.
 */
static gint configure_event (GtkWidget *widget, GdkEventConfigure *event)
{
    /* --- Free background if we created it --- */
    if (pixmap) {
        gdk_pixmap_unref (pixmap);
    }

    /* --- Create a new pixmap with new size --- */
    pixmap = gdk_pixmap_new (widget->window,
                widget->allocation.width,
                widget->allocation.height,
                -1);

    return TRUE;
}


/*
 * GotFocus
 *
 * Called when the window gets the focus.  This is necessary
 * because the keys repeat themeselves and block out other
 * keys when pressed at the same time.  The only way around
 * this is to disable that behavior and handle the key "up"
 * and "down" ourselves.  Of course, this change is global
 * across all applications, so we should set it only when we
 * have the focus.
 */
static gint GotFocus (GtkWidget *widget, gpointer data)
{
    gdk_key_repeat_disable ();
}

/*
 * LosFocus
 *
 * See GotFocus.  Because the change is global, we should
 * restore the repeat so that other applications aren't
 * messed up.
 */
```

```
static gint LostFocus (GtkWidget *widget, gpointer data)
{
    gdk_key_repeat_restore ();
}


/*
 * KeyPress
 *
 * Hey, a keypress.  Add the key to the list of
 * keys currently being pressed.
 */
static gint KeyPress (GtkWidget *widget, GdkEventKey *event)
{

    AddKey (event);
}

/*
 * KeyRelease
 *
 * Hey, they let go of a keypress.  Remove the
 * key from the list of keys that is currently
 * being pressed.
 */
static gint KeyRelease (GtkWidget *widget, GdkEventKey *event)
{
    RemoveKey (event);
}

/*
 * expose_event
 *
 * When the window is exposed to the viewer or
 * the gdk_widget_draw routine is called, this
 * routine is called.  Copies the background pixmap
 * to the window.
 */
gint expose_event (GtkWidget *widget, GdkEventExpose *event)
{

    /* --- Copy pixmap to the window --- */
```

```
    gdk_draw_pixmap (widget->window,
          widget->style->fg_gc[GTK_WIDGET_STATE (widget)],
          pixmap,
          event->area.x, event->area.y,
          event->area.x, event->area.y,
          event->area.width, event->area.height);

    return FALSE;
}


/*
 * GetPen
 *
 * Get a pen using the GdkColor passed in.  The "pen"
 * (just a GdkGC) is created and returned ready for
 * use.
 */
GdkGC *GetPen (GdkColor *c)
{
    GdkGC *gc;

    /* --- Create a gc --- */
    gc = gdk_gc_new (pixmap);

    /* --- Set the foreground to the color --- */
    gdk_gc_set_foreground (gc, c);

    /* --- Return it. --- */
    return (gc);
}


/*
 * NewColor
 *
 * Create and allocate a GdkColor with the color
 * specified in the parameter list.
 */
GdkColor *NewColor (long red, long green, long blue)
{
    /* --- Get the color --- */
```

```
    GdkColor *c = (GdkColor *) g_malloc (sizeof (GdkColor));

    /* ---Fill it in. --- */
    c->red = red;
    c->green = green;
    c->blue = blue;

    gdk_color_alloc (gdk_colormap_get_system (), c);

    return (c);
}


/*
 * quit
 *
 * Get out of the application
 */
void quit ()
{
    gtk_exit (0);
}



/*
 * main
 *
 * Program begins here.
 */
int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *drawing_area;
    GtkWidget *vbox;

    StartGame ();

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    /*---Don't allow window to be resized---*/
```

```
gtk_window_set_policy (GTK_WINDOW(WINDOW), FALSE, FALSE, TRUE);
vbox = gtk_hbox_new (FALSE, 0);
gtk_container_add (GTK_CONTAINER (window), vbox);
gtk_widget_show (vbox);

gtk_signal_connect (GTK_OBJECT (window), "destroy",
            GTK_SIGNAL_FUNC (quit), NULL);

/* --- Create the drawing area --- */
drawing_area = gtk_drawing_area_new ();
gtk_drawing_area_size (GTK_DRAWING_AREA (drawing_area), 400, GetGameHeight ());
gtk_box_pack_start (GTK_BOX (vbox), drawing_area, TRUE, TRUE, 0);

gtk_widget_show (drawing_area);

gtk_widget_set_events (window, GDK_KEY_RELEASE_MASK);

/* --- Signals used to handle backing pixmap --- */
gtk_signal_connect (GTK_OBJECT (drawing_area), "expose_event",
            (GtkSignalFunc) expose_event, NULL);
gtk_signal_connect (GTK_OBJECT (drawing_area), "configure_event",
            (GtkSignalFunc) configure_event, NULL);

/* --- Focus signals --- */
gtk_signal_connect (GTK_OBJECT (window), "focus_in_event",
            (GtkSignalFunc) GotFocus, NULL);
gtk_signal_connect (GTK_OBJECT (window), "focus_out_event",
            (GtkSignalFunc) LostFocus, NULL);

/* --- Key press signals --- */
gtk_signal_connect (GTK_OBJECT (window), "key_press_event",
            (GtkSignalFunc) KeyPress, NULL);
gtk_signal_connect (GTK_OBJECT (window), "key_release_event",
            (GtkSignalFunc) KeyRelease, NULL);

/* --- Show the window --- */
gtk_widget_show (window);

/* --- Repaint every second --- */
gtk_timeout_add (100, Repaint, drawing_area);

LoadImages (window);
```

```
    /* --- Call gtk_main loop --- */
    gtk_main();

    return 0;
}



/*
 * GetWidthHeight
 *
 * Get the height and width of the pixmap from the xpm data.
 */
void GetWidthHeight(gchar **xpm, int *width, int *height)
{

        sscanf(xpm[0], "%d %d", width, height);
}
```

13.2.3.8　keys.c

文件 keys.c 保持对按下的键的跟踪，并根据按下的键调用相应的移动和开火函数。

```
/*
 * Auth: Eric Harlow
 * File: keys.c
 * Developing Linux Applications
 */

#include <gtk/gtk.h>
#include <gdk/gdkkeysyms.h>
#include "defender.h"


/*
 * Movement
 */
int keyLeft = 0;
int keyRight = 0;
int keyUp = 0;
int keyDown = 0;
```

```
/*
 * Firing
 */
int keyFire = 0;

/*
 * AddKey
 *
 * Add the key to the keys that are being kept
 * track of.  We only keep track of a few of the keys,
 * and we ignore the rest.
 */
void AddKey (GdkEventKey * event)
{
switch (event->keyval) {

        /* ---Left arrow--- */
        case GDK_Left:
            keyLeft = TRUE;
            break;

        /* ---Right arrow--- */
        case GDK_Right:
            keyRight = TRUE;
            break;

        /* ---Up arrow--- */
        case GDK_Up:
            keyUp = TRUE;
            break;

        /* ---Down arrow--- */
        case GDK_Down:
            keyDown = TRUE;
            break;

        /* ---Space--- */
        case ' ':
            keyFire = TRUE;
            break;

        /* ---Ignore the rest--- */
```

```
            default:
                break;
        }
    }


/*
 * RemoveKey
 *
 * If we're keeping track of a key and the key
 * goes up, we flag it as not being pressed.
 */
void RemoveKey (GdkEventKey * event)
{
    switch (event->keyval) {

        case GDK_Left:
            keyLeft = FALSE;
            break;

        case GDK_Right:
            keyRight = FALSE;
            break;

        case GDK_Up:
            keyUp = FALSE;
            break;

        case GDK_Down:
            keyDown = FALSE;
            break;

        case ' ':
            keyFire = FALSE;
            break;

        default:
            break;
    }
}
```

```
/*
 * HandleKeysPressed
 *
 * When it comes time to move everyone, this
 * routine is called to move/fire based on the
 * keys that are currently pressed.
 */
void HandleKeysPressed ()
{

    /*
     * Try and move in each direction
     */
    if (keyLeft) {
        HeroMove (MOVE_LEFT);
    }

    if (keyRight) {
        HeroMove (MOVE_RIGHT);
    }

    if (keyUp) {
        HeroMove (MOVE_UP);
    }

    if (keyDown) {
        HeroMove (MOVE_DOWN);
    }

    /*
     * Try and fire.
     *
     * Of course, once we fire, we clear the flag so
     * the player has to hit the key for multiple shots.
     * Otherwise they could just hold down the key.
     */
    if (keyFire) {

        /* ---Zap--- */
        HeroFire ();

        /* ---Oh my, key is good for only one shot--- */
```

```
        keyFire = FALSE;
    }
}
```

13.2.3.9　defender.c

本文件是用于所有游戏规则和绘图的主要模块，它不处理与界面有关的问题。它只绘制构成游戏空间的绘图区。

```
/*
 * File: defender.c
 * Auth: Eric Harlow
 *
 * Sort of like the 1980s game Defender, but not a
 * complete version.
 *
 */

#include <gtk/gtk.h>
#include "defender.h"
#include "defproto.h"
#include "mutant.h"
#include "math.h"

/*
 * How many people and aliens should the game start
 * out with?
 */
#define START_ALIENS 10
#define START_PEOPLE 10

/*
 * These are for the acceleration of the player/hero, the
 * effect of friction to slow him down, and the maximum
 * velocity that can be attained.
 */
#define FRICTION .5
#define MAX_SPEED 16
#define ACCELERATION 3.5

/*
 * Of course, the laser has to be configurable too.  The
```

```
 * laser beam has a speed (how fast it travels out) and
 * a length (primarily for show).
 */
#define LASER_LENGTH 60
#define LASER_SPEED 60

/*
 * Range of the game.
 */
#define X_RANGE 2000

/*
 * How high the mountain peaks are.
 */
#define MAX_PEAK 150

/*
 * The number of mountain peaks generated.
 */
#define NUM_PEAKS 10

/*
 * --- Trigger happiness
 *
 * The mutants are much more trigger happy than the landers
 * are.
 */
#define LANDER_TRIGGER_PCT 3
#define MUTANT_TRIGGER_PCT 6

/*
 * During an explosion, we use eight sprites spiraling away
 * from where the item exploded.  These are the eight
 * directions.
 */
int x_exp[] = {1, 1, 0, -1, -1, -1, 0, 1};
int y_exp[] = {0, 1, 1, 1, 0, -1, -1, -1};



/*
 * How big is the radar screen?
```

```
 */
#define RADAR_WIDTH 200
#define RADAR_HEIGHT 50

/*
 * How big are other things in the screen?
 */
#define GAME_HEIGHT 220
#define BOTTOM_HEIGHT 30
#define PERSON_HEIGHT (RADAR_HEIGHT + GAME_HEIGHT + 7)

/*
 * Colors used for drawing
 */
GdkGC *penGreen = NULL;
GdkGC *penWhite = NULL;
GdkGC *penPurple = NULL;
GdkGC *penRed = NULL;

/*
 * Variables used in calculations
 */
int nShipAdjustment;
int nRelativeAdjustment;
int nScreenWidth;

/*
 * Define the sprites for the cast of characters.
 */
typSprite sprite_man[1]     = {{xpm_man, NULL, NULL, 0, 0}};
typSprite sprite_ship1[1]   = {{xpm_ship1, NULL, NULL, 0, 0}};
typSprite sprite_ship2[1]   = {{xpm_ship2, NULL, NULL, 0, 0}};
typSprite sprite_lander[1]  = {{xpm_lander, NULL, NULL, 0, 0}};
typSprite sprite_mutant[1]  = {{xpm_mutant, NULL, NULL, 0, 0}};
typSprite sprite_missile[1] = {{xpm_missile, NULL, NULL, 0, 0}};

/*
 * The ship could be pointing left or right (sprite_ship1,
 * sprite_ship2) - however, sprite_ship is what's used to
 * draw it.  If the ship changes direction, then this
 * should also be adjusted.
 */
```

```
typSprite * sprite_ship = sprite_ship1;


/*
 * List of mountain peaks.
 */
GList  * terrainList = NULL;

/*
 * All the units in the game.
 */
GList * unitList = NULL;

/*
 * The hero.
 */
typUnit * hero = NULL;


/*
 * Prototypes.
 */
GdkColor * NewColor (long red, long green, long blue);
GdkGC * GetPen (GdkColor * c);
void GetWidthHeight (gchar ** xpm, int * width, int * height);
void DisplaySprite (GtkWidget * drawing_area, typSprite * sprite, int x, int y);


/*
 * UnitScreenX
 *
 * Convert the unit's X coordinates to screen coordinates
 * relative to the hero's coordinates.
 */
int UnitScreenX (typUnit * unit)
{
    int xPos;

    /* --- Adjust -x- if out of range --- */
    if (unit->x < 0) {
        unit->x += X_RANGE;
    } else if (unit->x > X_RANGE) {
```

```
        unit->x -= X_RANGE;
    }

    /* --- Make it relative --- */
    xPos = (int)(unit->x - nRelativeAdjustment);

    /* --- Readjust -x- if out of range --- */
    if (xPos < 0) xPos += X_RANGE;
    if (xPos > X_RANGE) xPos -= X_RANGE;

    return (xPos);
}


/*
 * ScreenX
 *
 * Take the game -x- value and convert it to the screen
 * -x- value.
 */
int ScreenX(int x)
{
    int xPos;

    /* --- Adjust if out of range --- */
    if (x < 0) {

        x += X_RANGE;

    } else if (x > X_RANGE) {

        x -= X_RANGE;
    }

    /* --- Make it absolute --- */
    xPos = (int)(x - nRelativeAdjustment);

    /* --- Readajust if out of range --- */
    if (xPos < (-(X_RANGE - nScreenWidth) / 2)) {

        xPos += X_RANGE;
```

```
    } else if (xPos > ((X_RANGE - nScreenWidth) / 2)) {

        xPos -= X_RANGE;
    }

    return (xPos);
}


/*
 * GameX
 *
 * Take a screen -x- value and convert it to the
 * game coordinate -x- value.
 */
int GameX (int x)
{
    int xPos;

    /* --- Relative to the hero --- */
    xPos = (int) (x + nRelativeAdjustment);

    /* --- Make sure it's not out of range --- */
    if (xPos < 0) xPos += X_RANGE;
    if (xPos > X_RANGE) xPos -= X_RANGE;

    return (xPos);
}


/*
 * Move
 *
 * Move a unit in the X direction by vx (velocity in x) and
 * in the Y direction by vy.
 */
void Move (typUnit *unit)
{

    /* --- Move the unit --- */
    unit->y += unit->vy;
    unit->x += unit->vx;
```

```
    /* --- But keep it within the world --- */
    if (unit->x < 0) unit->x += X_RANGE;
    if (unit->x > X_RANGE) unit->x -= X_RANGE;
}


/*
 * LoadPixmaps
 *
 * Load the image into the sprite.
 */
void LoadPixmaps (GtkWidget *widget, typSprite *sprites)
{

    /* --- Create a pixmap from the xpm data --- */
    sprites->pixmap = gdk_pixmap_create_from_xpm_d (
                                widget->window,
                                &sprites->mask,
                                NULL,
                                sprites->xpm_data);

    /* --- Get the width and height --- */
    GetWidthHeight (sprites->xpm_data,
                    &sprites->width,
                    &sprites->height);

}

/*
 * LoadImages
 *
 * Load up the images so that we can display them in the
 * game and configure the colors.
 */
void LoadImages (GtkWidget *window)
{
    /* --- Load up the images --- */
    LoadPixmaps (window, sprite_man);
    LoadPixmaps (window, sprite_ship1);
    LoadPixmaps (window, sprite_ship2);
    LoadPixmaps (window, sprite_lander);
```

```
    LoadPixmaps (window, sprite_mutant);
    LoadPixmaps (window, sprite_missile);

    /* --- Get the colors defined --- */
    penRed = GetPen (NewColor (0xffff, 0x8888, 0x8888));
    penGreen = GetPen (NewColor (0, 0xffff, 0));
    penPurple = GetPen (NewColor (0xffff, 0, 0xffff));
    penWhite = GetPen (NewColor (0xffff, 0xffff, 0xffff));
}


/*
 * CreateHero
 *
 * Create the typUnit for the player and initialize
 * it with the player settings.
 */
typUnit *CreateHero ()
{
    /* --- Allocate the memory --- */
    hero = g_malloc (sizeof (typUnit));

    /* --- Initialize player information --- */
    hero->bDestroy = FALSE;
    hero->direction = 1;
    hero->type = HERO;
    hero->x = 0;
    hero->y = 150;
    hero->vx = 0;
    hero->vy = 0;
    hero->lockunit = NULL;

    /* --- Return object. --- */
    return (hero);
}


/*
 * HeroFire
 *
 * Our hero is opening fire!  Create a laser shot
 * and add it to the global list of units.
```

```
 */
void HeroFire()
{
    typUnit *laser;

    /* --- Create the laser --- */
    laser = (typUnit *) g_malloc(sizeof(typUnit));

    /* --- The direction is the same as the spaceship --- */
    laser->direction = hero->direction;

    laser->type = LASER;

    /*
     * Move the starting point of the laser in front of the
     * spaceship.
     */
    if (laser->direction > 0) {
        laser->x = hero->x + (sprite_ship->width / 2);
    } else {
        laser->x = hero->x - (sprite_ship->width / 2);
    }
    laser->y = hero->y + 4;

    laser->vx = LASER_SPEED * hero->direction;
    laser->vy = 0;
    laser->lockunit = NULL;
    laser->bDestroy = 0;

    /* --- Laser bolt lasts through two movements. --- */
    laser->life = 2;

    /* --- Add the laser to the list of units --- */
    unitList = g_list_append(unitList, laser);
}


/*
 * HeroMove
 *
 * Move the player in the direction specified.
 */
```

```
void HeroMove (int direction)
{
    switch (direction) {

        case MOVE_UP:
            hero->y -= 3;
            break;

        case MOVE_DOWN:
            hero->y += 3;
            break;

        case MOVE_LEFT:
            /*
             * Make sure the ship is pointing in the
             * correct direction.
             */
            sprite_ship = sprite_ship2;
            hero->direction = -1;

            /* --- Speed up the ship --- */
            hero->vx -= ACCELERATION;
            break;

        case MOVE_RIGHT:
            /*
             * Make sure the ship is pointing in the
             * correct direction.
             */
            sprite_ship = sprite_ship1;
            hero->direction = 1;

            /* --- Speed up the ship --- */
            hero->vx += ACCELERATION;
            break;
    }
}


/*
 * ApplyFriction
 *
```

```
 * Slow down gradually and make sure the player cannot go
 * into warp speed by holding down the acceleration.
 */
void ApplyFriction ()
{
    /* --- Slow ship down --- */
    if (hero->vx > FRICTION) {
        hero->vx -= FRICTION;
    } else if (hero->vx < -FRICTION) {
        hero->vx += FRICTION;
    } else {

        /* --- Speed less than friction, we stop --- */
        hero->vx = 0;
    }

    /* --- Don't let the maximum speed be exceeded --- */
    if (hero->vx > MAX_SPEED) hero->vx = MAX_SPEED;
    if (hero->vx < -MAX_SPEED) hero->vx = -MAX_SPEED;
}



/*
 * CreatePerson
 *
 * Create the little people on the planet.
 */
typUnit *CreatePerson ()
{
    typUnit *person;

    /* --- Allocate the memory --- */
    person = g_malloc (sizeof (typUnit));

    /* --- Initialize the person --- */
    person->bDestroy = FALSE;
    person->direction = 0;
    person->type = PERSON;
    person->x = rand () % X_RANGE;
    person->y = PERSON_HEIGHT;
    person->vx = 0;
    person->vy = 0;
```

```
    person->lockunit = NULL;

    return (person);
}


/*
 * PlacePeople
 *
 * Create people randomly and place them on the screen
 */
void PlacePeople ()
{
    int i;
    typUnit *person;

    /* --- Create all the little people --- */
    for (i = 0; i < START_PEOPLE; i++) {

        /* --- Create a person --- */
        person = CreatePerson ();

        /* --- Add it to the list of units --- */
        unitList = g_list_append (unitList, person);
    }
}


/*
 * CreateAlien
 *
 * Create an alien lander
 */
typUnit *CreateAlien ()
{
    typUnit *alien;

    /* --- Get the memory --- */
    alien = g_malloc (sizeof (typUnit));

    /* --- Initialize the structure --- */
```

```
    alien->bDestroy = FALSE;
    alien->pctFiring = LANDER_TRIGGER_PCT;
    alien->type = LANDER;
    alien->x = rand() % X_RANGE;
    alien->y = rand() % 50 + RADAR_HEIGHT;
    alien->vx = 0;
    alien->vy = 0;
    alien->lockunit = NULL;

    return (alien);
}



/*
 * CreateMissile
 *
 * Missiles are what the aliens fire.  They are given
 * a fixed direction and last a limited amount of time
 * before they disappear.
 */
typUnit *CreateMissile(typUnit *alien, typUnit *hero)
{
    float flength;
    typUnit *missile;

    /*---Allocate the memory---*/
    missile = (typUnit *) g_malloc(sizeof(typUnit));

    /*---Initialize the structure---*/
    missile->bDestroy = FALSE;
    missile->pctFiring = 0;
    missile->type = MISSILE;
    missile->x = alien->x;
    missile->y = alien->y;

    /*---Calculate missile velocity---*/
    missile->vx = (float) DistanceBetween(missile, hero) *
                          Direction(missile, hero);
    missile->vy = (float) (hero->y - alien->y);

    /*
     * Adjust missile velocity
```

```
     */
    flength = sqrt (missile->vx * missile->vx + missile->vy * missile->vy);
    if (flength < .1) flength = .1;
    flength /= 3;
    missile->vx /= flength;
    missile->vy /= flength;

    missile->lockunit = NULL;

    /*
     * --- This missile has a life; when it goes to zero,
     *     it gets destroyed.
     */
    missile->life = 60;

    return (missile);
}


/*
 * AddExplosion
 *
 * A unit was destroyed - we create an explosion at the
 * unit's location
 */
void AddExplosion (typUnit *unit)
{
    typUnit *frag;
    int i;

    /* --- Creating eight fragments --- */
    for (i = 0; i < 8; i++) {

        /* --- Create a fragment of the explosion --- */
        frag = (typUnit *) g_malloc (sizeof (typUnit));

        /* --- Initialize the fragment --- */
        frag->bDestroy = FALSE;
        frag->pctFiring = 0;
        frag->type = EXPLOSION;
        frag->x = unit->x;
        frag->y = unit->y;
```

```
        /* --- Moves quite fast --- */
        frag->vx = x_exp[i] * 5;
        frag->vy = y_exp[i] * 5;

        frag->lockunit = NULL;

        /* --- Short life --- */
        frag->life = 20;

        /* --- Add it to the list of units --- */
        unitList = g_list_append (unitList, frag);
    }
}


/*
 * PlaceAliens
 *
 * Create the aliens.
 */
void PlaceAliens ()
{
    int i;
    typUnit *alien;

    /* --- Create each of the aliens --- */
    for (i = 0; i < START_ALIENS; i++) {

        /* --- Create one alien --- */
        alien = CreateAlien ();

        /* --- Add it to the units list --- */
        unitList = g_list_append (unitList, alien);
    }
}


/*
 * DistanceBetween
 *
 * Calculate the distance between two units - but
```

```
 * only use the X direction to calculate the distance.
 * The distance is not just the difference between
 * the units X coordinates, but because the world
 * wraps around, we have to consider the fact that
 * the distance between 1 and X_RANGE - 1 is not
 * (X_RANGE -1) - 1
 */
int DistanceBetween (typUnit *u1, typUnit *u2)
{
    int nDistance;
    int nDistance2;

    /* --- Figure out which is greater --- */
    if (u1->x < u2->x) {

        /* --- Calculate distance both ways --- */
        nDistance = u2->x - u1->x;
        nDistance2 = X_RANGE + u1->x - u2->x;

    } else {
        /* --- Calculate distance both ways --- */
        nDistance = u1->x - u2->x;
        nDistance2 = X_RANGE + u2->x - u1->x;
    }

    /* --- Pick smaller of the distances --- */
    return ((nDistance < nDistance2) ? nDistance : nDistance2);
}

/*
 * Direction
 *
 * What is the -x- directory between the two units?
 * It's either going to be -1, 1, or 0.
 */
int Direction (typUnit *u1, typUnit *u2)
{
    int nDistance;
    int nDistance2;

    if (u1->x < u2->x) {
```

```
        /* --- Distance each way --- */
        nDistance = u2->x - u1->x;
        nDistance2 = X_RANGE + u1->x - u2->x;

    } else {
        /* --- Distance each way --- */
        nDistance2 = u1->x - u2->x;
        nDistance = X_RANGE + u2->x - u1->x;
    }

    /* --- Direction depends on closer distance --- */
    return ((nDistance < nDistance2) ? 1 : -1);
}


/*
 * AttemptFiring
 *
 * Have the alien fire at the hero.
 */
void AttemptFiring (typUnit *unit)
{
    typUnit *missile;

    /* --- Close enough to shoot at --- */
    if (DistanceBetween (hero, unit) < (nScreenWidth / 2)) {

        /* --- Random chance - less than unit's chance --- */
        if ((rand () % 100) < unit->pctFiring) {

            /* --- Create the missle going after the hero --- */
            missile = CreateMissile (unit, hero);

            /* --- Add to the list of units --- */
            unitList = g_list_append (unitList, missile);
        }
    }
}


/*
```

```
 * AIModule
 *
 * Contains the logic for each of the units to move.
 * Some units like landers are looking for people to
 * pick up.  Mutuants hunt down the player.  Missiles
 * just go until they die, etc.
 */
void AIModule (typUnit *unit)
{
    typUnit *tmp;
    int bestdist = 50000;
    typUnit *closest = NULL;
    GList *node;

    /*
     * Alien lander AI logic
     */
    if (unit->type == LANDER) {

        /* --- if the alien's locked on a person --- */
        if (unit->lockunit) {

            /* --- Move it up --- */
            closest = unit->lockunit;
            unit->y -= .5;
            closest->y -= .5;

            /* --- If alien's made it to the top --- */
            if (unit->y - (sprite_lander[0].height / 2) < RADAR_HEIGHT) {

                /* --- Assimilate human! --- */
                unit->y = RADAR_HEIGHT + (sprite_lander[0].height / 2);
                unit->type = MUTANT;
                unit->pctFiring = MUTANT_TRIGGER_PCT;
                unit->lockunit = NULL;
                closest->bDestroy = TRUE;
            }

            return;
        }

        /* --- Any people nearby to snatch? --- */
```

```
for (node = unitList; node; node = node->next) {

    tmp = (typUnit *) node->data;
    if (tmp->type == PERSON && tmp->lockunit == NULL) {

        /* --- Look for a closer person --- */
        if (DistanceBetween (unit, tmp) < bestdist) {
            closest = tmp;
            bestdist = DistanceBetween (unit, tmp);
        }
    }
}

/* --- We're locked onto a target --- */
if (bestdist <= 1) {

    /* --- Scootch it over a bit --- */
    unit->vx = 0;
    unit->x = closest->x;

    /*
     * --- Check for a lock... ---
     */

    if ((unit->y + (sprite_lander[0].height / 2) + .8) <
        (closest->y - (sprite_man[0].height / 2))) {

        /* --- Come down on it. --- */
        unit->y += .5;

    } else if ((unit->y + (sprite_lander[0].height / 2)) >
        (closest->y - (sprite_man[0].height / 2))) {

        unit->y -= .5;
    } else {

        /* --- Lock it in --- */
        unit->lockunit = closest;
        closest->lockunit = unit;
        closest->life = 20;
    }
```

```
        /* --- Anything in reasonable range? --- */
        } else if (bestdist < 20) {

            /* --- Move towards it --- */
            unit->vx = Direction (unit, closest);
            unit->x += unit->vx;
        } else {

            /*
             * --- Nothing nearby.  Move in a random direction.
             */
            if (unit->vx == 0) {
                if ((rand() % 2) == 0) {
                    unit->vx = 1;
                } else {
                    unit->vx = -1;
                }
            }
            unit->x += unit->vx;
        }

        /*
         * See if there is anything worth shooting at.
         */
        AttemptFiring (unit);

    /*
     * Mutant AI logic
     */
    } else if (unit->type == MUTANT) {

        /*
         * --- Let's go crazy.  Mutant moves almost randomly, yet
         *  .  slowly towards the player.
         */
        unit->vx = Direction (unit, hero) * ((rand() % 4) + 1);
        unit->vy = rand() % 5 - 2;

        /*
         * If the hero is within smelling distance, move towards
         * player in the -y- direction.
         */
```

```
        if (DistanceBetween (unit, hero) < 200) {
            if (unit->y < hero->y) unit->vy++;
            if (unit->y > hero->y) unit->vy--;
        }

        /* --- Finally move the unit --- */
        Move (unit);

        /* --- Let the mutant attempt firing --- */
        AttemptFiring (unit);

    /*
     * --- Missiles and explosions
     */
    } else if ((unit->type == MISSILE) ||
               (unit->type == EXPLOSION)) {

        /* --- These have a life.  Decrement it. --- */
        unit->life--;

        /* --- Move it. --- */
        Move (unit);

        /* --- When it reaches zero, destroy it --- */
        if (unit->life <= 0) {
            unit->bDestroy = TRUE;
        }

    /*
     * Person AI
     */
    } else if (unit->type == PERSON) {

        /*
         * Only time person moves by itself is when it's falling
         * from the sky after the alien carrying it has been shot.
         */
        if (unit->lockunit == NULL && unit->y < PERSON_HEIGHT) {

            /* --- Move it down --- */
            unit->y += 2;
        }
```

```
    }
}


/*
 * GetSprite
 *
 * Get the sprite for the unit.
 */
typSprite *GetSprite(typUnit *unit)
{
    typSprite *sprite;

    /* ---Get sprite--- */
    switch (unit->type) {

        case HERO:
            sprite = sprite_ship;
            break;

        case PERSON:
            sprite = sprite_man;
            break;

        case LANDER:
            sprite = sprite_lander;
            break;

        case MUTANT:
            sprite = sprite_mutant;
            break;

        case MISSILE:
        case EXPLOSION:
            sprite = sprite_missile;
            break;

        default:
            sprite = NULL;
            break;
    }
    return (sprite);
```

```
}

/*
 * AnyoneBetween
 *
 * Is any unit between these coordinates?  Used by the
 * laser when computing if anything has been hit.
 */
typUnit *AnyoneBetween (int x1, int y1, int x2, int y2)
{
    GList *node;
    typUnit *unit;
    typUnit *closestUnit = NULL;
    int closestX;
    typSprite *sprite;
    int screenX;

    /* --- Check each unit --- */
    for (node = unitList; node; node = node->next) {

        unit = (typUnit *) node->data;

        /* --- If it's a bad guy --- */
        if ((unit->type == LANDER) ||
            (unit->type == PERSON) ||
            (unit->type == MUTANT)) {

            /* --- Get sprite and screen position --- */
            screenX = UnitScreenX (unit);
            sprite = GetSprite (unit);

            /* --- If we have a sprite --- */
            if (sprite) {

                /* --- If in range using -x- --- */
                if ((screenX >= x1 && screenX <= x2) ||
                    (screenX <= x1 && screenX >= x2)) {

                    /* --- And in range using -y- --- */
                    if ((unit->y - (sprite->height / 2) < y1) &&
                        unit->y + (sprite->height / 2) > y1) {
```

```
                        /*
                          * Haven't found a unit or this one
                          * is closer.
                          */
                        if ((closestUnit == NULL) ||
                            (abs (x1 - screenX) < abs (x1 - closestX))) {

                            /* --- This is closest so far --- */
                            closestUnit = unit;
                            closestX = screenX;
                        }
                    }
                }
            }
        }

        return (closestUnit);
}


/*
 * UnitTop
 *
 * Calculate the maximum height of the unit based on the
 * radar screen and the sprite size.
 */
int UnitTop (typUnit *unit)
{
    typSprite *sprite;

    /* --- Get the sprite --- */
    sprite = GetSprite (unit);

    /* --- Add 1/2 sprite size to radar size. --- */
    return (RADAR_HEIGHT + (sprite[0].height / 2));
}


/*
 * UnitBottom
 *
```

```
 * Calculate how low the unit can appear on the screen.
 * It's partially based on the unit's size.
 */
int UnitBottom (typUnit * unit)
{
    typSprite * sprite;

    sprite = GetSprite (unit);
    return (GetGameHeight () - (sprite[0].height / 2));
}


void AdjustSpriteHeight (typUnit * unit)
{
    typSprite * sprite;
    int nTop;
    int nBottom;

    /* --- Get the sprite for the unit --- */
    sprite = GetSprite (unit);
    if (sprite == NULL) return;

    /* --- Calculate the top and bottom range of unit --- */
    nTop = UnitTop (unit);
    nBottom = UnitBottom (unit);

    /* --- Don't let them go too high or too low --- */
    if (unit->y < nTop) {
        unit->y = nTop;
    } else if (unit->y > nBottom) {
        unit->y = nBottom;
    }
}


/*
 * DisplayOtherUnits
 *
 * Display all the units on the screen. First, we need
 * to move the units to their new positions.
 * Some of this is done in the AI module.
 *
```

```
 */
void DisplayOtherUnits (GdkPixmap *pixmap, GtkWidget *drawing_area)
{
    typUnit *unit;
    typUnit *unitHit;
    GList *node;
    int xPos;
    int xPosEnd;
    typSprite *sprite;

    /* --- Each unit in the list --- */
    for (node = unitList; node; node = node->next) {

        /* --- Get the unit --- */
        unit = (typUnit *) node->data;

        /*
         * --- Run the AI module on it to move it ---
         */
        AIModule (unit);

        /*
         * If the unit was destroyed by the AI,
         * don't draw the unit.
         */
        if (unit->bDestroy) {
            continue;
        }

        /*
         * If there's no sprite for the unit,
         * we can't draw it now, can we?
         */
        sprite = GetSprite (unit);
        if (sprite == NULL) continue;

        /* --- Where on the screen is it going? --- */
        xPos = UnitScreenX (unit);

        /* --- Make sure unit doesn't go out of bounds --- */
        AdjustSpriteHeight (unit);
```

```
    /* --- Finally draw unit --- */
    DisplaySprite (drawing_area, sprite,
                   (int) (xPos - sprite[0].width / 2),
                   (int) (unit->y - sprite[0].height / 2));
}

/*
 * --- Once everyone is painted, fire the lasers.
 */

for (node = unitList; node; node = node->next) {

    unit = (typUnit *) node->data;

    /* --- If this is a laser --- */
    if (unit->type == LASER) {

        /* --- Get starting and ending positions --- */
        xPos = ScreenX ((int) unit->x);
        xPosEnd = xPos + LASER_LENGTH * unit->direction;

        /* --- See if anything was hit --- */
        unitHit = AnyoneBetween ((int) xPos, (int) unit->y,
                                 (int) xPosEnd, (int) unit->y);
        if (unitHit) {

            /* --- Something was hit --- */

            /* --- Laser shot only goes this far --- */
            xPosEnd = UnitScreenX (unitHit);

            /* --- Destroy the unit --- */
            unitHit->bDestroy = TRUE;
            unit->bDestroy = TRUE;

            /* --- Special effects of destruction --- */
            AddExplosion (unitHit);
        }

        /* --- Draw the laser --- */
        gdk_draw_line (pixmap, penWhite,
                       xPos, unit->y,
```

```
                        xPosEnd,
                        unit->y);

            /* --- Get real coordinates of laser --- */
            unit->x = GameX (xPosEnd);

            /* --- If laser has gone too far... --- */
            if (DistanceBetween (unit, hero) > nScreenWidth / 2) {

                /* --- Destroy it --- */
                unit->bDestroy = TRUE;
            }
        }
    }
}



/*
 * Terrain functions
 *
 */


/*
 * MountainCompare
 *
 * Compare function for sorting the mountain peaks.
 */
gint MountainCompare (typMountain *m1, typMountain *m2)
{
    return (m1->start.x - m2->start.x);
}


/*
 * AddMountain
 *
 * Add a mountain peak to the list we use to keep track
 * of the mountain peaks.
 */
GList *AddMountain (GList *mountainList, int peakx, int peaky)
```

```
{
    typMountain * node;

    node = (typMountain * ) g_malloc (sizeof (typMountain));
    node->start.x = peakx - peaky;
    node->start.y = 0;
    node->peak.x = peakx;
    node->peak.y = peaky;
    node->end.x = peakx + peaky;
    node->end.y = 0;

    return (g_list_insert_sorted (mountainList, node, MountainCompare));
}


/*
 * AddPoint
 *
 * Add a mountain peak at (x, y)
 */
GList *AddPoint (GList * terrainList, int x, int y)
{
    typPoint * p;

    /* --- Allocate the memory --- */
    p = (typPoint * ) g_malloc (sizeof (typPoint));

    /* --- Initialize the point --- */
    p->x = x;
    p->y = y;

    /* --- Add the point to the list --- */
    terrainList = g_list_append (terrainList, p);

    return (terrainList);
}


/*
 * GenerateTerrain
 *
 * Create random mountain peaks that are used to generate
```

```
 * the background terrain.
 */
void GenerateTerrain ()
{
    int peakx;
    int peaky;
    GList *mountainList = NULL;
    GList *node;
    typMountain *prevMountain;
    typMountain *mtn;
    int i;

    /* --- Compute the peaks --- */
    for (i = 0; i < NUM_PEAKS; i++) {
        peakx = rand () % X_RANGE;
        peaky = rand () % MAX_PEAK;

        mountainList = AddMountain (mountainList, peakx, peaky);
    }

    prevMountain = NULL;

    terrainList = AddPoint (terrainList, 0, 0);

    /* --- Compute the lines based on the peaks --- */
    for (node = mountainList; node; node = node->next) {

        mtn = (typMountain *) node->data;

        /* --- First mountain --- */
        if (prevMountain == NULL) {
            terrainList = AddPoint (terrainList, mtn->start.x, mtn->start.y);
            terrainList = AddPoint (terrainList, mtn->peak.x, mtn->peak.y);
            prevMountain = mtn;

        /* --- Don't cross paths --- */
        } else if (prevMountain->end.x < mtn->start.x) {

            terrainList = AddPoint (terrainList,
                                    prevMountain->end.x,
                                    prevMountain->end.y);
            terrainList = AddPoint (terrainList, mtn->start.x, mtn->start.y);
```

```
            terrainList = AddPoint (terrainList, mtn->peak.x, mtn->peak.y);
            prevMountain = mtn;

        /* --- Previous mountain eats this one --- */
        } else if (prevMountain->end.x > mtn->end.x) {

            /* --- Do nothing yet --- */
        } else {

            /* --- Mountains intersect --- */
            terrainList = AddPoint (terrainList,
                                    (prevMountain->end.x + mtn->start.x) / 2,
                                    (prevMountain->end.x - mtn->start.x) / 2);
            terrainList = AddPoint (terrainList, mtn->peak.x, mtn->peak.y);
            prevMountain = mtn;
        }
    }
    terrainList = AddPoint (terrainList,
                                    prevMountain->end.x,
                                    prevMountain->end.y);
    terrainList = AddPoint (terrainList, X_RANGE, 0);
}



void StartGame ()
{
    unitList = NULL;

    /* --- Create our hero's ship --- */
    hero = CreateHero ();

    /* --- Generate the map --- */
    GenerateTerrain ();

    /* --- Place the people --- */
    PlacePeople ();

    /* --- Place aliens --- */
    PlaceAliens ();
}
```

```
/*
 * DrawMountainSlope
 *
 * Draw the mountains in the background of the game.
 */
void DrawMountainSlope (GdkPixmap *pixmap,
                        typPoint *lastPt,
                        typPoint *pt,
                        int nTop,
                        int nBottom)
{
    int x1;
    int x2;
    int nStart;
    int nEnd;
    int nHeight;

    nHeight = nBottom - nTop;
    nStart = hero->x - (nScreenWidth / 2);
    nEnd = hero->x + (nScreenWidth / 2);

    x1 = ScreenX (lastPt->x);
    x2 = ScreenX (pt->x);

    if ((x2 < 0) || (x1 > nScreenWidth)) {

        /* ---Do nothing--- */

    } else {

        /* ---Draw point--- */
        gdk_draw_line (pixmap, penWhite,
                   ScreenX (lastPt->x),
                   (int) (nBottom - ((lastPt->y * nHeight) / MAX_PEAK)),
                   ScreenX (pt->x),
                   (int) (nBottom - ((pt->y * nHeight) / MAX_PEAK)));
    }
}


void DrawMountains (GdkPixmap *pixmap,
                    GtkWidget *drawing_area,
```

```
                    int nTop, int nBottom)
{
    typPoint *lastPt;
    typPoint *pt;
    GList *node;

    lastPt = NULL;

    /* --- Where is the viewpoint from? --- */
    for (node = terrainList; node; node = node->next) {

        /* --- Get the point. --- */
        pt = (typPoint *) node->data;

        if (lastPt) {

            DrawMountainSlope (pixmap, lastPt, pt, nTop, nBottom);
        }
        lastPt = pt;
    }
}


void CalculateAdjustments (GtkWidget *drawing_area)
{
    nShipAdjustment = (drawing_area->allocation.width / 2);
    nRelativeAdjustment = hero->x - nShipAdjustment;
}


/*
 * DrawAllUnits
 *
 * Draw all the units
 */
void DrawAllUnits (GdkPixmap *pixmap,
                   GtkWidget *drawing_area)
{
    /*
     * Move and display the hero
     */
    ApplyFriction ();
```

```
    Move (hero);

    /* --- Keep him in-bounds. --- */
    AdjustSpriteHeight (hero);

    DisplaySprite (drawing_area, sprite_ship,
                   nShipAdjustment - (sprite_ship->width / 2),
                   (int) hero->y - (sprite_ship->height / 2));
    /*
     * Move and display everyone else
     */
    DisplayOtherUnits (pixmap, drawing_area);
}



/*
 * DrawRadar
 *
 * Draw all the units on the radar screen.  Of course, the
 * radar screen has to be drawn and then the units have to
 * have their points converted to "radar" coordinates and
 * little dots are drawn on the screen in different colors
 * to reflect the type of each unit in the game.
 */
void DrawRadar (GdkPixmap *pixmap, GtkWidget *drawing_area)
{
    int   nLeft;
    GList *node;
    typUnit *unit;
    int x, y;
    int nMin;
    int newx, newy;

    /* --- Get radar coordinates --- */
    nLeft = (drawing_area->allocation.width - RADAR_WIDTH) / 2;
    nMin = hero->x - (X_RANGE / 2);

    /* --- Clear out the rectangle --- */
    gdk_draw_rectangle (pixmap, drawing_area->style->white_gc,
                        FALSE,
                        nLeft, 0, RADAR_WIDTH, RADAR_HEIGHT);
```

```
/*
 * --- Display all the units
 */
for (node = unitList; node; node = node->next) {

    unit = (typUnit *) node->data;
    x = unit->x;
    y = unit->y;

    if (x > hero->x + (X_RANGE / 2)) {

        x -= X_RANGE;

    } else if (x < hero->x - (X_RANGE / 2)) {

        x += X_RANGE;
    }

    /* --- Convert -x- to radar coordinates --- */
    newx = (x - nMin) * RADAR_WIDTH / X_RANGE;

    /* --- Convert -y- to radar coordinates --- */
    newy = ((y - RADAR_HEIGHT) * (RADAR_HEIGHT - 6) / GAME_HEIGHT) + 2;

    switch (unit->type) {

        case PERSON:
            gdk_draw_rectangle (pixmap, penPurple,
                    TRUE, nLeft + newx-1, newy-1, 2, 2);
            break;

        case LANDER:
            gdk_draw_rectangle (pixmap, penGreen,
                    TRUE, nLeft + newx-1, newy-1, 2, 2);
            break;

        case MUTANT:
            gdk_draw_rectangle (pixmap, penRed,
                    TRUE, nLeft + newx-1, newy-1, 2, 2);
            break;

        case MISSILE:
```

```
            case EXPLOSION:
                gdk_draw_rectangle (pixmap, penWhite,
                        TRUE, nLeft + newx, newy, 1, 1);
                break;
        }
    }

    /*
     * --- Put the hero on the screen too.
     */

    /* --- Convert -x- to radar coordinates --- */
    newx = (hero->x - nMin) * RADAR_WIDTH / X_RANGE;

    /* --- Convert -y- to radar coordinates --- */
    newy = ((hero->y - RADAR_HEIGHT) * (RADAR_HEIGHT - 6) / GAME_HEIGHT) + 2;

    gdk_draw_rectangle (pixmap, penWhite, TRUE,
                        nLeft + newx-1, newy-1, 2, 2);
}


/*
 * DrawScreen
 *
 * This does a lot of work.  It draws the background and
 * all the units, as well as the radar.  This routine is
 * essentially the main loop that gets called by the
 * timer every update frequency.
 */
void DrawScreen (GdkPixmap *pixmap, GtkWidget *drawing_area)
{
    /* --- Move player based on keys pressed --- */
    HandleKeysPressed ();

    /* --- Get the screen width --- */
    nScreenWidth = drawing_area->allocation.width;

    /* --- Figure out the offset of the player --- */
    CalculateAdjustments (drawing_area);

    /* --- Clear pixmap (background image) --- */
```

```
    gdk_draw_rectangle (pixmap,
                drawing_area->style->black_gc,
                TRUE,
                0, 0,
                drawing_area->allocation.width,
                drawing_area->allocation.height);

    /* --- Draw top border and radar screen --- */
    gdk_draw_line (pixmap, drawing_area->style->white_gc,
                0, RADAR_HEIGHT,
                drawing_area->allocation.width,
                RADAR_HEIGHT);

    /* --- Oh those high peaks --- */
    DrawMountains (pixmap, drawing_area,
                drawing_area->allocation.height - 65,
                drawing_area->allocation.height - BOTTOM_HEIGHT);

    /* --- Draw the characters --- */
    DrawAllUnits (pixmap, drawing_area);

    /* --- Draw the units on the radar --- */
    DrawRadar (pixmap, drawing_area);

    /* --- Look for collisions --- */
    CollisionCheck ();

    /* --- Clean up those that got destroyed --- */
    FreeDestroyedUnits ();
}


/*
 * CollisionCheck
 *
 * Hero suffers a collision?  Check for a collision
 * but don't really do anything.  This isn't really
 * a game, but rather a demo on how this could be done.
 */
void CollisionCheck ()
{
    GList *node;
```

```
    typSprite *sprite;
    int hero_x;
    int unit_x;
    typUnit *unit;

    /* --- See if anything collides with the hero. --- */
    for (node = unitList; node; node = node->next) {

        /* --- Get the unit --- */
        unit = (typUnit *) node->data;

        /* --- Get the unit's sprite --- */
        sprite = GetSprite (unit);

        if (sprite == NULL) continue;

        /* --- Eliminate non-collisions --- */
        if (hero->y + (sprite_ship->height / 2) <
            unit->y - (sprite->height / 2)) continue;

        if (hero->y - (sprite_ship->height / 2) >
            unit->y + (sprite->height / 2)) continue;

        hero_x = UnitScreenX (hero);
        unit_x = UnitScreenX (unit);

        if (hero_x + (sprite_ship->width / 2) <
            unit_x - (sprite->width / 2)) continue;

        if (hero_x - (sprite_ship->width / 2) >
            unit_x + (sprite->width / 2)) continue;

        /* --- Hero collides here since everything failed. --- */
    }
}


/*
 * GetGameHeight
 *
 * Get the height of the game.
 */
```

```
int GetGameHeight ()
{

    return (RADAR_HEIGHT + GAME_HEIGHT + BOTTOM_HEIGHT);
}


/*
 * FreeDestroyedUnits
 *
 * The units are just marked as being destroyed.  This
 * routine goes through the list and removes all the units
 * that are marked as destroyed and frees up any memory
 * they're using.
 */
void FreeDestroyedUnits ()
{
    GList *node;
    GList *next;
    typUnit *unit;

    node = unitList;
    while (node) {

        next = node->next;

        unit = (typUnit *) node->data;

        /* --- If it's to be destroyed --- */
        if (unit->bDestroy) {

            /* --- Remove from the list of units --- */
            unitList = g_list_remove (unitList, unit);

            /* --- If unit was locked to another --- */
            if (unit->lockunit) {

                /* --- Remove lock to this unit --- */
                unit->lockunit->lockunit = NULL;
            }
            g_free (unit);
        }
```

```
            node = next;
        }
    }
```

13.3 小结

你能够利用 GTK + 和 GDK 建立活动的图像和电视游戏。即使在比较慢的机器上,在 GTK + 中的基本电视游戏的性能也是不错的。更复杂的电视游戏需要更快的机器,也许需要一个更快的库。

第四部分　扩充 GTK +

第 14 章　树、列表和标记
第 15 章　创建自己的构件

第 14 章　树、列表和标记

GTKTree、GTKCList、GTKNotebook 比普通的录入构件更使人感兴趣。这 3 种构件显示出使用 GTK + 开发应用程序的优势。同时，这些构件还提供面向对象的界面，使得用构件开发应用程序更加有效和便捷。虽然，在 GKT + 中其他构件也十分有效，但这虽我们将要讨论的 3 种构件将更加有用。

14.1　树构件

树构件(GTKTree)显示分层的信息：一组目录，或是显示你的双亲、双亲的双亲等的家族树。树允许显示包含树中所有项目的整个层次结构，或者将子项归并到父项下。

图 14.1 表示用 GTKTree 和列表框构件显示目录和文件的例子。在窗口的左边列出了目录树。在树中，每个目录的子目录作为可以归并到父节点的叶节点加以显示。当一个目录被选中后，它所包含的文件被显示在窗口的右边。现在让我们看看如何用树构件编写这个程序。

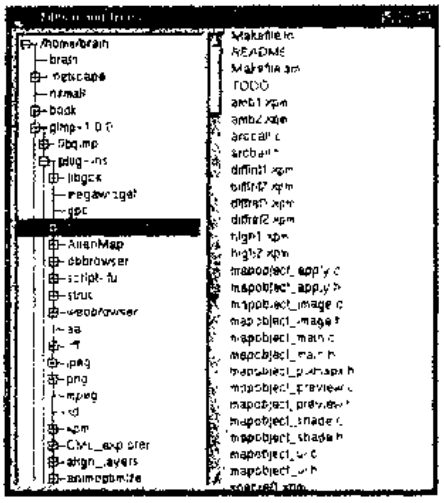


图 14.1　树构件的使用

14.1.1　树的建立

使用 gtk _ tree _ new 函数来建立树(GtkTree)构件。刚刚建立的树是空的，而且需要加以填充。以叶节点名称作为参数，使用 gtk _ tree _ item _ new _ with _ label 函数来建立叶节点。建立叶节点后，需要将它指定给父节点。叶节点的父节点可以是 GtkTree 构件或者是另一个

叶节点。现在你可以建立树构件,并将它的顶层节点加进去。

```
/* --- Create the tree --- */
tree = gtk_tree_new();

/* --- Create a leaf node --- */
leaf = gtk_tree_item_new_with_label ("Top level leaf!");

/* --- Add item to the tree --- */
gtk_tree_append (GTK_TREE (tree), leaf);

/* --- Don't forget to make tree and leaves visible --- */
gtk_widget_show (leaf);
gtk_widget_show (tree);
```

现在,可以将一些叶节点加到树上去。但是,在加入构成叶节点的数据需要用几步。第一步,用 gtk_tree_new 函数建立一个新树。第二步,用 gtk_tree_item_set_subtree 函数标记这个树是一个叶节点,并给它指定一个父节点。第三步,使用前面我们在树中加入叶节点的函数在叶节点中加入它的组成部分。

```
/* --- Create a new tree item --- */
subtree = gtk_tree_new ();

/* --- Make the leaf a parent to the subtree --- */
gtk_tree_item_set_subtree (GTK_TREE_ITEM (leaf), subtree);

/* --- Create a new leaf --- */
new_leaf = gtk_tree_item_new_with_label ("leaf");

/* --- Add new leaf to the subtree --- */
gtk_tree_append (GTK_TREE (subtree), new_leaf);

/* --- Make new leaf visible --- */
gtk_widget_show (new_leaf);
```

函数 gtk_tree_append 是唯一可以在顶层节点中加入项目的函数。我们可以使用 gtk_tree_prepend 函数在一个树的开始插入项目,或者使用 gtk_tree_insert 函数在树的任何部位插入项目。

```
/* --- Add an item at the beginning of a tree --- */
gtk_tree_prepend (GTK_TREE (tree), leaf);

/* --- Add an item at beginning using insert --- */
```

```
gtk_tree_insert (GTK_TREE (tree), leaf, 0);

/* --- Add to end of tree using insert --- */
gtk_tree_insert (GTK_TREE (tree), leaf, -1);

/* --- Insert after the second element --- */
gtk_tree_insert (GTK_TREE (tree), leaf, 2);
```

可以使用 gtk_tree_remove_item 函数删除树中的项目。当删除一个项目后，它所包含的所有叶节点都将被删除。

```
/* --- Remove the leaf from the tree --- */
gtk_tree_remove_item (tree, leaf)
```

可以使用 gtk_tree_clear_items 函数来清除树中的叶结点。指定开始和结束的位置后，其间的项目将会被删除。让我们看看如何删除所有的叶节点：

```
/* --- Remove all children from tree --- */
gtk_tree_clear_items (tree,0,-1);
```

14.1.2　树的信号

树构件有一些自己的信号，“SELECTION_CHANGED”、“SELECT_CHILD”和“UNSELECT_CHILD”信号被发送到树构件以指示被选项目的状态。树的项目接收“COLLAPSE_TREE”和“EXPAND_TREE”信号。

14.1.3　建立文件浏览器

利用前面掌握的内容，现在我们来建立文件浏览器。在应用程序窗口中包含并列的树构件和列表框构件。第一步，让我们用主函数来建立应用程序窗口。

```
/*
 * main
 *
 * Program begins here
 */
int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *table;

    /* --- GTK initialization --- */
    gtk_init (&argc, &argv);
```

```
    /* --- Create the top window --- */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- Give the window a title --- */
    gtk_window_set_title (GTK_WINDOW (window), "Files using trees");

    /* --- Set the window size --- */
    gtk_widget_set_usize (window, 250, 250);

    /* --- You should always remember to connect the delete_event
     *      to the main window.
     */
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (delete_event), NULL);

    gtk_widget_show (window);

    /* --- Create the tree  --- */
    CreateTree (window);

    gtk_main ();

    exit (0);
}
```

CreateTree 函数建立横向组合框,使得可以将树构件和列表框构件并列地放置在里面。因为将树中的项目展开时,有可能超出窗口可以显示的高度。所以将树构件放置在带有垂直滚动条的窗口内。这样,用户就可以使用滚动条来观察其中的项目。getcwd 函数从系统中获得当前工作目录,并将它作为树构件的根显示。将目录信息传递到 CreateSubTree 函数,使用它来显示所有目录。

```
/*
 * CreateTree
 *
 * Create the tree that shows the file structure.
 *
 * window  parent window
 */
static void CreateTree (GtkWidget *window)
{
    char buffer[MAX_PATH];
    GtkWidget *box1;
```

```
GtkWidget *scrolled_win;
GtkWidget *tree;
GtkWidget *leaf;
GtkWidget *item1, *item2, *item3;
GtkWidget *tree1, *tree2, *tree3;
GtkWidget *item;

/* --- Vertical box --- */
box1 = gtk_hbox_new (FALSE, 0);
gtk_container_add (GTK_CONTAINER (window), box1);
gtk_widget_show (box1);

/* --- Create scrolled window for the tree --- */
scrolled_win = gtk_scrolled_window_new (NULL, NULL);
gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolled_win),
                                GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);
gtk_box_pack_start (GTK_BOX (box1), scrolled_win, TRUE, TRUE, 0);
gtk_widget_set_usize (scrolled_win, 250, 250);
gtk_widget_show (scrolled_win);

/*
 * --- Create root tree widget
 */
tree = gtk_tree_new();

/*
 * --- Create list box
 */
listbox = gtk_list_new ();
gtk_widget_set_usize (listbox, 250, 250);
gtk_box_pack_start (GTK_BOX (box1), listbox, TRUE, TRUE, 0);
gtk_widget_show (listbox);

/* --- Add the tree to the scrolled window --- */
gtk_container_add (GTK_CONTAINER (scrolled_win), tree);

/* --- Make it visible --- */
gtk_widget_show (tree);

/*
 * --- Create root tree item widget (current directory)
 */
```

```
    leaf = gtk_tree_item_new_with_label (
                    getcwd (buffer, sizeof (buffer)));

    /* --- Add item to the tree --- */
    gtk_tree_append (GTK_TREE (tree), leaf);

    /* --- Make it visible --- */
    gtk_widget_show (leaf);

    /* --- Create a subtree under this item --- */
    CreateSubtree (getcwd (buffer, sizeof (buffer)),
                    getcwd (buffer, sizeof (buffer)), leaf);

    /* --- Make window visible --- */
    gtk_widget_show (window);
}
```

CreateSubtree 函数获得当前目录中的子目录,这些目录成为当前目录的子节点。例如,book 目录可能包含目录 ch1、ch2 和 ch3。在这种情况下,称 book 节点包含 ch1、ch2 和 ch3 三个子节点。函数 opendir 显示任一目录中包含的子目录和文件。使用 IsDirectory 函数判断目录中的项目是目录还是文件。如果是目录,它又被递归地传递给 CreateSubtree,由 CreateSubtree 自动找出有关的子目录。每个节点都有"select"信号的回呼函数。所以,当你点击一个目录时,将在列表框中显示这一目录包含的所有文件。

```
/*
 * CreateSubtree
 *
 * Adds directories into the tree control and puts in
 * an event handler so that when an item is selected
 * in the tree, the list box will be populated.
 *
 * szPath - path to add the files from
 */
static void CreateSubtree (char * szPath, char * szDir, GtkWidget * item)
{
    DIR * directory;
    struct dirent * dirEntry;
    struct stat fileStatus;
    GtkWidget * item_subtree = NULL;
    GtkWidget * item_new;
    char outDir[MAX_PATH];
    int nSubDirectories = 0;
```

```
/* --- Read current directory --- */
directory = opendir (szPath);

/* --- While we are reading through the directory --- */
while (dirEntry = readdir (directory)) {

    /* --- Don't count these as valid directories for display --- */
    if (!strcmp (dirEntry->d_name, "..") ||
        !strcmp (dirEntry->d_name, ".")) {

        /* --- Ignore these directories (".." and ".") --- */
    } else {

        /* --- Create full path --- */
        sprintf (buffer, "%s/%s", szPath, dirEntry->d_name);

        /* --- If this is a directory --- */
        if (IsDirectory (buffer)) {

            if (item_subtree == NULL) {

                /* --- Create a new tree item --- */
                item_subtree = gtk_tree_new ();

                /* --- Add the item to the tree --- */
                gtk_tree_item_set_subtree (GTK_TREE_ITEM (item),
                                           item_subtree);

            }

            /* --- Create an entry for the file --- */
            item_new = gtk_tree_item_new_with_label (dirEntry->d_name);

            /* --- Add item to the tree --- */
            gtk_tree_append (GTK_TREE (item_subtree), item_new);

            /* --- Load all of its elements into the tree --- */
            CreateSubtree (buffer, dirEntry->d_name, item_new);

            /* --- Make it visible --- */
            gtk_widget_show (item_new);
```

```
                /* --- Notify when an item is selected --- */
                gtk_signal_connect (GTK_OBJECT (item_new),
                        "select",
                        GTK_SIGNAL_FUNC (select_item),
                        g_strdup (buffer));

            }
        }
    }

    /* --- All done --- */
    closedir (directory);

    gtk_widget_show (item);
```

IsDirectory 函数获取文件/目录的名称,并用 stat 确定这一项目是文件还是目录

```
IsDirectory

 * Checks to see whether the path is a directory or just a file.

 * buffer - full path to check.

 * Returns (TRUE) if path is a directory
 */
gint IsDirectory (gchar *buffer)
{
    struct stat buf;

    if (stat (buffer, &buf) < 0)
    {
        /* -- Error - ignored --- */
        return (FALSE);
    }

    /* --- Return whether it is a directory -- */
    return (S_ISDIR (buf.st_mode));
}
```

当在树构件中点击一个项目时,将调用 Select_item 函数。这个函数显示一条信息,

调用 PopulateListbox 函数显示这个目录中包含的文件列表。

```
void select_item (GtkWidget *widget, gpointer data)
{
    printf ("item selected %s\n", (char *) data);

    PopulateListbox ((char *) data);
}
```

不需要对 PopulateListbox 的代码进行解释，只要给定一个目录，它就会在列表框中显示目录中的所有的文件名。

```
/*
 * PopulateListbox
 *
 * Adds the files for the directory into the list box.
 * Only adds files.
 *
 * szPath - path to add the files from
 */
static void PopulateListbox (char *szPath)
{
    DIR *directory;
    struct dirent *dirEntry;
    struct stat fileStatus;
    char buffer[MAX_PATH];

    /* --- Clear the list box --- */
    gtk_list_clear_items (GTK_LIST (listbox),
                          0,
                          g_list_length (GTK_LIST (listbox)->children));

    /* --- Read current directory --- */
    directory = opendir (szPath);

    /* --- While we are reading through the directory --- */
    while (dirEntry = readdir (directory)) {

        /* --- Don't count these as valid directories for display --- */
        if (!strcmp (dirEntry->d_name, "..") ||
            !strcmp (dirEntry->d_name, ".")) {

            /* --- Ignore these directories ("." and "..") --- */
```

```
        } else {

            /* --- Create full path --- */
            sprintf (buffer, "%s/%s", szPath, dirEntry->d_name);

            /* --- If this not directory --- */
            if (!IsDirectory (buffer)) {

                /* --- Add file to the list box --- */
                AddListItem (listbox, dirEntry->d_name);
            }
        }
    }

    /* --- All done --- */
    closedir (directory);
}
```

AddListItem 是经过简化的将字符串加到列表框中的函数。

```
/*
 * AddListItem
 *
 * Add an item to a list box
 *
 * listbox - list box to add the item to
 * text - text string to display in the list box
 */
void AddListItem (GtkWidget *listbox, char *sText)
{
    GtkWidget *item;

    /* --- Create list item from data --- */
    item = gtk_list_item_new_with_label (sText);

    /* --- Add item to list box --- */
    gtk_container_add (GTK_CONTAINER (listbox), item);

    /* --- Make it visible --- */
    gtk_widget_show (item);
}
```

以上得到的是文件浏览器应用程序。它以当前工作目录为起点，不涉及任何上层目录。它读文件系统的结构，从当前工作目录开始显示文件系统树。点击任何目录名称，将显示该目录中的所有文件。点击目录名称旁边的“+”或“-”符号可以扩展或压缩目录。

14.2　笔记本构件

笔记本构件(GtkNotebook)使得将信息显示在几页中成为可能。通过点击代表每一页的标记(tab)就能将这一页显示在最前面。每一个标记上都有说明，以便用户辨别是哪一页。标记可以出现在页的上方、下方、左方或右方。

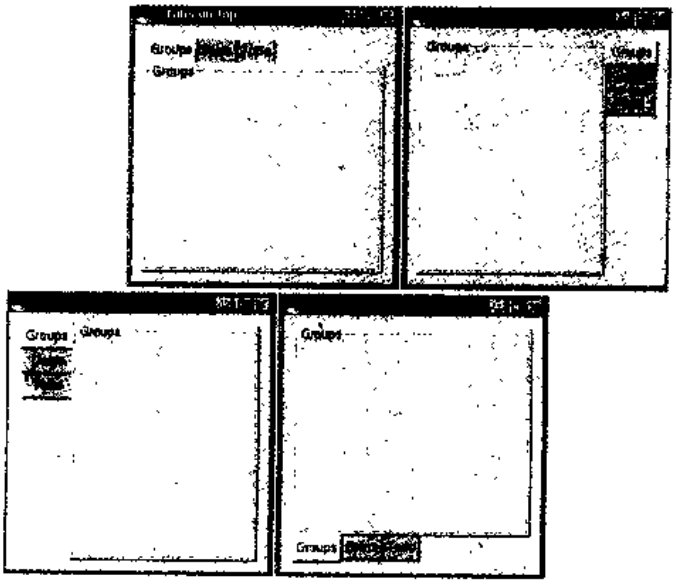


图 14.2　笔记本构件

使用 gtk _ notebook _ new 函数建立笔记本构件。使用 gtk _ notebook _ set _ tab _ pos 函数建立标记和设置标记的位置(GTK _ POS _ TOP, GTK _ POS _ BOTTOM, GTK _ POS _ LEFT, GTK _ POS _ RIGHT)。

```
/* --- Create the notebook --- */
notebook = gtk_notebook_new ();

/* --- Put the tabs along the top of the notebook --- */
gtk_notebook_set_tab_pos (GTK_NOTEBOOK (notebook), GTK_POS_TOP);
```

14.2.1　增加和删除页

如果没有在笔记本构件中放入一些页，它是不能使用的。我们可以使用函数 gtk _ notebook _ append _ page 在几个页的后面再加入页。gtk _ notebook _ prepend _ page 函数正好和它

相反,它在最前面加入一页。而 gtk _ notebook _ insert _ page 函数则可以在任何位置插入一页。我们应该在建立页之前先建立构件,并将构件作为参数传递给这些函数。这一步通常包括建立标记的标号和某种类型的容器。这样,可以将其他构件加到容器内。一般使用 gtk _ notebook _ remove _ page 函数来删除页。

```
/* --- Create label and container for page --- */
label = gtk_label_new ("First tab");
container = gtk_frame_new ("Public information");

/* --- Add a page to the end of the notebook --- */
gtk_notebook_append_page (notebook, container, label);

/* --- Create label and container for page --- */
label = gtk_label_new ("Last tab");
container = gtk_frame_new ("Secret information");

/* --- Add a page to the end of the notebook --- */
gtk_notebook_prepend_page (notebook, container, label);

/* --- Create label and container for page --- */
label = gtk_label_new ("Middle tab");
container = gtk_frame_new ("Quasi secret information");

/* --- Add a page to the end of the notebook --- */
gtk_notebook_insert_page (notebook, container, label);
```

14.2.2 处理页

可以用 gtk _ notebook _ current _ page 函数来获得当前页。用 gtk _ notebook _ set _ page 函数来设置当前页。用 gtk _ notebook _ next _ page 和 gtk _ notebook _ prev _ page 函数来向前和向后翻过一页。用 gtk _ notebook _ set _ show _ tabs 函数来隐藏或显示笔记本的标记。

```
/* --- Get the current page --- */
nPage = gtk_notebook_current_page (notebook);

/* --- Go to the next page --- */
gtk_notebook_set_page (notebook, nPage + 1);

/* --- Go to the previous page --- */
gtk_notebook_prev_page (notebook);

/* --- Go to the next page (again) --- */
```

```
gtk_notebook_next_page (notebook);

/* --- Hide the tabs from the user --- */
gtk_notebook_set_show_tabs (notebook, FALSE);
```

隐藏标记一开始的确是令人费解的。因为这样一来用户将无法在笔记本中通过点击标记来切换到其他页。但是，在一些场合下需要由应用程序来控制当前应该显示哪一页。这种情况类似微软的应用程序 wizard 引导用户通过一个复杂的过程。这个过程可能需要用户通过 10 个步骤，每次将一个步骤显示在一页上通常是比较方便的。当用户成功地完成一个步骤后，应用程序接着显示下一页。

在本章最后的程序中包含若干页，用来说明在应用程序中应该如何使用笔记本构件。

14.3　列表构件

列表(GtkCList)构件使你能在单一构件内浏览表格信息。列表构件内可以包含任意行和列，而且列的宽度可以设置成任意大小。列表构件经常用于显示数据库信息，其中，在列表构件中每一列相当于数据库中的一个字段，而每一行则相当于数据库中的一行。

一旦用固定的列数建立了列表构件，它所包含的列数就不可以改变。如果要改变列表构件内的的列数，需要删除列表构件后重新建立。可以以需要的列数作为参数使用 gtk_clist_new 函数建立列表构件。也可以使用 gtk_clist_new_with_titles 函数来建立它。可以使用 gtk_clist_set_column_title 函数来设置或者改变列的名称。

```
/* --- Create a table with 3 columns --- */
clist = gtk_clist_new(3);

/* --- Give each column a title --- */
gtk_clist_set_column_title (GTK_CLIST, 0, "Id");
gtk_clist_set_column_title (GTK_CLIST, 1, "Name");
gtk_clist_set_column_title (GTK_CLIST, 2, "Address");
```

上面的一段代码可以简化如下：

```
/* --- Define the titles --- */
char *szTitles[] = {"Id","Name","Address"};

/* --- Create the clist with titles all in one shot --- */
Clist = gtk_clist_new_with_titles (3, szTitles);
```

14.3.1　将数据加入列表构件

可以使用 gtk_clist_append 函数在列表的最后加入数据。传递给 gtk_clist_append 函数的参数是 GtkCList 和字符串数组。数组元素的个数应该和列表构件的列数保持一致。

```
/* --- Adding a row of static data --- */
char *szData = {"0123","Eric","123 Main St."};

/* --- Add the row of data to the clist --- */
gtk_clist_append (GTK_CLIST (clist), szData);
```

可以用 gtk _ clist _ insert 函数向列表中的任意位置插入数据,但我们需要在参数中指出将数据插入哪一行:

```
/* --- Insert a row of static data --- */
char *szData = {"0123","Eric","123 Main St."};

/* --- Insert the data after row 10 --- */
gtk_clist_insert (GTK_CLIST (clist), 10, szData);
```

将正文加入列表构件后,可以用 gtk _ clist _ set _ text 函数修改它。函数 gtk _ clist _ get _ text 使用指向字符串的指针(char **)作为参数,并获得指定行与列的数据。如果成功地填充了指针,函数返回非 0 值。因为不能直接存取列表中的数据,我们必须使用这种方法修改数据。

```
char *data

/* --- change the data in an existing row/colom --- */
gtk_clist_set_text (GTK_CLIST (clist), 3, 1, "joe");

/* --- Get the text --- */
gtk_clist_get_text (GTK_CLIST (clist), 3, 1, &data);
```

14.3.2 删除行

可以以被删除行的索引值作为参数,使用 gtk _ clist _ remove 函数从列表构件中删除一行。但是,如果我们想删除列表构件内的所有项,使用 gtk _ clist _ clear 函数会更加快捷。

```
/* --- Remove first row --- */
gtk_clist_remove (CLIST (clist), 0);

/* --- Remove all the items from the list --- */
gtk_clist_clear (CLIST (clist));
```

14.3.3 提高插入和删除的速度

正像可以将大量数据加到正文构件中一样,列表构件主要是为了显示大量的数据,特别是数据库文件。为了提高在列表构件中插入和修改数据的速度,我们可以像正文构件一样"冻结"列表构件,直到完成插入或修改数据为止。例如:使用 gtk _ clist _ freeze 函数冻结列表构

件，直到函数 gtk _ clist _ thaw 被调用时再“解冻”列表。

```
/* --- Freeze the widget --- */
gtk_clist_freeze (CLIST (clist));

/* --- Do some processing --- */

/* --- Unfreeze the list --- */
gtk_clist_thaw (CLIST (clist));
```

14.3.4　标题栏的特征

列表构件的标题栏具有可以在应用程序中改变的特征。例如，你可以决定是否显示标题栏。函数 gtk _ clist _ column _ titles _ hide 可以隐藏标题栏。同时，可以使用 gtk _ clist _ column _ titles _ show 函数来显示标题栏。函数 gtk _ clist _ set _ column _ widget 允许使用任何构件代替标题栏。这是在标题栏中显示图像的好方法。

```
/* --- Hide the title bars --- */
gtk_clist_column_titles_hide (GTK_CLIST (clist));

/* --- Make the titles visible --- */
gtk_clist_column_titles_show (GTK_CLIST (clist));

/* --- Put a previously created pixmap in as the title --- */
gtk_clist_set_column_widget (GTK_CLIST (clist), 1, pixmap_widget);
```

14.3.5　行与列参数

当建立列表构件后，需要手工调整列的宽度。一般来说，列表构件并不知道列真正需要多宽，它只是根据标题栏的宽度去猜想列的宽度。因此你需要使用函数 gtk _ clist _ set _ column _ width 来设置列的宽度。同样，可以使用 gtk _ clist _ set _ column _ width 函数来设置行的高度。但是，如果你不改变字体或增加大的图形，设置行的高度是不必要的。

```
/* --- Set the width of the column to 100 pixels --- */
gtk_clist_set_column_width (GTK_CLIST (clist), 0, 100);

/* --- Change the row height for the images --- */
gtk_clist_set_row_height (GTK_CLIST (clist), 25);
```

同样可以调整每一列中的数据位置，它可以左、右对齐或居中。所用的函数是 gtk _ clist _ set _ column _ justification，它使用构件、列和 GtkJustification 作为参数。它的有效值是 GTK _ JUSTIFY _ LEFT、GTK _ JUSTIFY _ RIGHT、GTK _ JUSTIFY _ CENTER 和 GTK _ JUSTIFY _ FILL。

14.3.6 图表

到此为止,只讨论了显示文本信息。有时浏览一行一行的正文和数字是十分枯燥的。因此列表构件也支持在行中插入像元图。可以使用 gtk _ clist _ set _ pixmap 函数插入像元图,使用 gtk _ clist _ get _ pixmapx 函数检查像元图。需要注意的是:当用插入操作或者附加操作加入正文数据时,应该将显示像元图的列设置为 NULL,否则,像元图将不会被显示。

```
/* ---Give the item a pixmap--- */
gtk_clist_set_pixmap (GTK_CLIST (clist), row, col, pix, mask);

/* ---Take a look at the existing pixmap--- */
gtk_clist_get_pixmap (GTK_CLIST (clist), row, col, &pix, &mask);
```

14.4 建立应用程序

在下面的应用程序中使用了笔记本构件和列表构件,它用来显示 Apache Web 服务器的日志文件。将日志文件分为 3 个笔记本页。第 1 页显示一天中每小时的通信流量(见图 14.3)。这样,你就可以判断在通信流量的高峰期间带宽是否充裕。第 2 页显示从使用日志文件起每天的通信流量。它使你了解通信流量的趋势——是什么原因造成通信流量的变化。

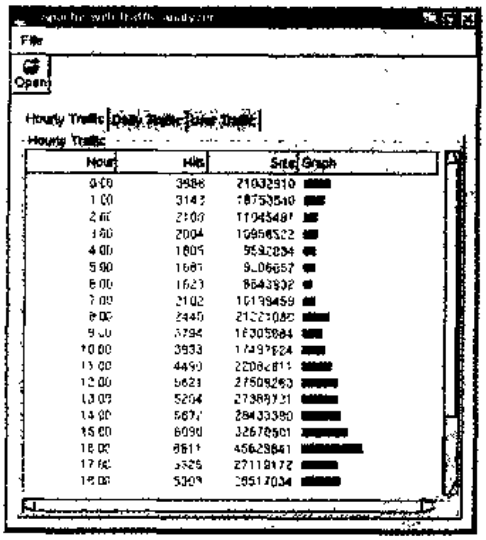


图 14.3 每小时的通信流量

第 3 页显示用户产生的通信流量。这个应用程序将主页作为单独的用户页。

每个列表构件使用像元图来显示信息。这使得用户可以迅速的发现问题所在。从图形中找出问题比从一行行数字中去找要快得多。

这个项目的许多文件以前都用过。提高代码的重用性是这里讨论的题目之一。文件 misc.c、progressbar.c 和 filesel.c 都在前面讨论过。对 Frontend.c 略作改动以适合不同的界面。在这个应用程序中的重要文件是日志分析程序(parselog.c)、GtkNotebook/clist 显示(Notebook.c)和像元图产生程序(bitmaps.c)。

在表中显示的像元图非常独特,它没有使用静态的像元图数据,面是动态建立的。像元图是由一个子程序根据显示的水平条的宽度产生的。水平的条形图容易自动产生,它比建立静态的像元图数据更为灵活。我们可以用改变产生像元图数据的子程序很容易地改变图形分辨率。如果我们使用静态图像,可能需要花费不少时间编辑构成条形图的可能像元图组合。

14.4.1　Logtypes.h

Logtypes.h 文件定义了在应用程序中使用的数据结构。每种结构都是为了放置特定的数据使用的。

TypeTimeStamp 保存访问 Web 页的日期和时间。

```
typedef struct {

   int year;
   int month;
   int day;
   int hours;
   int minutes;
   int seconds;

} typTimeStamp;
```

type Hit 结构保存所有与访问 Wob 站点有关的信息。

```
typedef struct {

    char * sIp;
    char * sUser;
    char * sDate;
    char * sCmd;
    char * sURL;
    int nResult;
    int nSize;
    typTimeStamp date;

} typHit;
```

type Stat 结构保存所有访问信息以及对 URL 传送的字节。

```
    typedef struct {

    char * sURL;
    long nHits;
    long nSize;

} typStat;
```

type Date 结构跟踪用于分析的日期。

```
typedef struct {

   int year;
   int month;
   int day;

} typDate;
```

type Date info 记录保存访问信息以及特定日期传送的字节。

```
typedef struct {

    long nHits;
    long nSize;
    typDate * date;

} typDateInfo;
```

14.4.2 Parselog.c

对每个被显示数据都需要读入和加以综合。信息被保存在树中,所以对信息的查找和提取是很快的。访问日志包含许多行数据,而你需要快速访问用于更新的信息。从访问日志读入每一个记录,对它进行分析,并作为一次访问信息保存。然后将它送给各个利用访问信息进行综合的函数。例如,如果访问信息与特定的用户页和特定的日期有关,则将这一访问信息加到该用户的综合信息和日期综合信息中。当分析日志文件的工作完成后,列表构件就用综合信息显示日志文件。

```
/*
 * File: parselog.c
 * Auth: Eric Harlow
 *
```

```
 * Parse an Apache log file and summarize information.
 */

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>
#include <gtk/gtk.h>
#include "logtypes.h"


/*
 * Define data for the storage of hourly information.
 */
long timeOfDayHits[24];
long timeOfDaySize[24];

/*
 * Define trees to store the different types of data.
 */
GTree * fileTree = NULL;
GTree * dateTree = NULL;
GTree * userTree = NULL;

void ParseLine (char * buffer);

#define MONTHS 12
char * sValidMonths[] = { "Jan","Feb", "Mar","Apr","May","Jun",
                          "Jul","Aug","Sep","Oct","Nov","Dec" };


/*
 * Date2Str
 *
 * Convert the date to a string.
 * date - date to convert
 * buffer - char * big enough to hold the date/time
 */
void Date2Str (typTimeStamp * date, char * buffer)
{
```

```
    sprintf (buffer, "%d/%d/%d %d:%d:%d",
                                date->month,
                                date->day,
                                date->year,
                                date->hours,
                                date->minutes,
                                date->seconds);
}


/*
 * ConvertDate
 *
 * Convert the date from a format of 30/Feb/1999:11:42:23
 * which is how the date was stored in my Apache log files.
 */
void ConvertDate (char *sDate, typTimeStamp *date)
{
    char sMonth[33];
    int i;

    /* --- Break down the date into its components --- */
    sscanf (sDate, "%d/%3c/%d:%d:%d:%d", &date->day,
                                         sMonth,
                                         &date->year,
                                         &date->hours,
                                         &date->minutes,
                                         &date->seconds);

    /* --- Convert the string date into a numeric --- */
    for (i = 0; i < MONTHS; i++) {

        /* --- Is this the month? --- */
        if (!strncasecmp (sValidMonths[i], sMonth, strlen (sValidMonths[i]))) {
            date->month = i + 1;
            break;
        }
    }
}


/*
```

```
 * ParseLog
 *
 * Parse the log file and organize the data in the log file into
 * a format that can be interpreted by a user.
 *
 * sFile - File to read.
 */
void ParseLog (char * sFile)
{
    FILE * fp;
    char buffer[350];
    long nFileLen = 0;
    struct stat fileStatus;

    /* --- Alloc trees and init data --- */
    Init ();

    /* --- Get filename --- */
    stat (sFile, &fileStatus);
    nFileLen = fileStatus.st_size;

    /* --- Open le file <--- that's French, you know --- */
    fp = fopen (sFile, "r");

    /* --- Make sure we opened * something * --- */
    if (fp) {

        /* --- Show the progress bar --- */
        StartProgress ();

        /* --- While we have data --- */
        while (!feof (fp)) {

            /* --- Update the progress bar with information --- */
            UpdateProgress (ftell (fp), nFileLen);

            /* --- Read a line from the file --- */
            fgets (buffer, sizeof (buffer), fp);

            /* --- If not end of file --- */
            if (!feof (fp)) {
```

```
            /* --- Parse the data --- */
            ParseLine (buffer);
        }
    }

    /* --- Done - don't need progress bar --- */
    EndProgress ();

    /* --- Close the file --- */
    fclose (fp);
  }
}


/*
 * ParseLine
 *
 * Parse the line retrieved from the log file and break it
 * down into its components and update the stats being
 * kept.
 */
void ParseLine (char *buffer)
{

  char *sUnknown;
  char *sNull;
  char *sDate;
  char output[88];
  typHit hit;

  /* --- Get basic information --- */
  hit.sIp = strtok (buffer, " ");
  sUnknown = strtok (NULL, " ");
  hit.sUser = strtok (NULL, " ");

  /* --- Extract the date --- */
  sDate = strtok (NULL, "\");
  sDate++;

  /* --- Convert the date string --- */
  ConvertDate (sDate, &hit.date);
```

```
    /* --- Get file, size, status --- */
    sNull = strtok (NULL, "\"");
    hit.sCmd = strtok (NULL, ".");
    hit.nResult = atoi (strtok (NULL, " "));
    hit.nSize = atoi (strtok (NULL, " "));

    /* --- Page was an error --- */
    if (hit.nResult == 404) return;

    /* --- Convert back to string --- */
    Date2Str (&hit.date, output);

    sUnknown = strtok (hit.sCmd, " ");
    hit.sURL = strtok (NULL, " ");

    /* --- Valid URL --- */
    if (hit.sURL) {

        /* --- Update the stats --- */
        UpdateStats (&hit);
    }
}



/*
  * UpdateStats
  *
  * Update information about the "hit"
  * Currently keep stats on:
  *       The time of day that the hit occurred on.
  *       The traffic occurring on each of tha days.
  *       The traffic associated with user's Web sites.
  */
UpdateStats (typHit * hit)
{
    /* --- Update hourly stats --- */
    timeOfDayHits[hit->date.hours]++;
    timeOfDaySize[hit->date.hours] += hit->nSize;

    /* --- Update this information --- */
    TrackFiles (hit);
```

```
    TrackDates (hit);
    TrackUsers (hit);
}



/*
 * TrackUsers
 *
 * Track traffic to the user's sites.
 */
TrackUsers (typHit *hit)
{
    char *sURL;
    typStat *stat;
    char *sUser;

    /* --- Bad data --- */
    if (hit->sURL == NULL ||
strlen (hit->sURL) == 0) return;

    /* --- Get first part of the path --- */
    sUser = strtok (&hit->sURL[1], "/");

    if (sUser == NULL) return;

    /* --- If it's a user site --- */
    if (sUser[0] == '~') {

        /* --- Get the name --- */
        sUser = &sUser[1];

    /* --- If ~ was encoded as %7E --- */
    } else if (!strcmp (sUser, "%7E")) {

        /* --- Get the name --- */
        sUser = &sUser[3];
    } else {

        /* --- Not a user, belongs to main site --- */
        sUser = "*ROOT";
    }
```

```
    /* --- See if the user exists in the tree --- */
    stat = g_tree_lookup (userTree, sUser);

    /* --- If user does not exist --- */
    if (stat == NULL) {

        /* --- Make space for the user --- */
        stat = g_malloc (sizeof (typStat));

        /* --- Populate fields --- */
        stat->sURL = strdup (sUser);
        stat->nHits = 0;
        stat->nSize = 0;

        /* --- Insert user into tree --- */
        g_tree_insert (userTree, stat->sURL, stat);
    }

    /* --- Update hit count for user --- */
    stat->nHits ++;
    stat->nSize += hit->nSize;
}


/*
 * TrackFiles
 *
 * Keep track of hit counts on individual files
 */
TrackFiles (typHit *hit)
{
    char *sURL;
    typStat *stat;

    /* --- Look up the URL in the tree --- */
    stat = g_tree_lookup (fileTree, hit->sURL);

    /* --- If the URL is not found --- */
    if (stat == NULL) {

        /* --- Create a node for the URL --- */
```

```
        stat = g_malloc (sizeof (typStat));

        /* --- Populate the node --- */
        stat->sURL = strdup (hit->sURL);
        stat->nHits = 0;
        stat->nSize = 0;

        /* --- Add the node to the tree --- */
        g_tree_insert (fileTree, stat->sURL, stat);
    }

    /* --- Update the file hit count --- */
    stat->nHits ++;
    stat->nSize = hit->nSize;
}

/*
 * TrackDates
 *
 * Track the traffic that occurs on any given date.
 */
TrackDates (typHit * hit)
{
    char * sURL;
    typStat * stat;
    typDate date;
    typDate * newdate;
    typDateInfo * info;

    /* --- Get the date of the hit --- */
    date.year = hit->date.year;
    date.month = hit->date.month;
    date.day = hit->date.day;

    /* --- Look up the date --- */
    info = g_tree_lookup (dateTree, &date);

    /* --- If no data for the date? --- */
    if (info == NULL) {

        /* --- Create field to hold date information --- */
```

```
        info = g_malloc (sizeof (typDateInfo));
        newdate = g_malloc (sizeof (typDate));

        /* --- Populate fields --- */
        *newdate = date;

        info->nHits = 0;
        info->nSize = 0;
        info->date = newdate;

        /* --- Add date info to the tree --- */
        g_tree_insert (dateTree, newdate, info);
    }

    /* --- Update date hit count --- */
    info->nHits ++;
    info->nSize += hit->nSize;
}


/*
 * CompareStrings
 *
 * Function to compare two strings for a tree callback.
 * Could have just passed strcmp into the tree function
 * but may want to change the comparision in the future.
 */
gint CompareStrings (gpointer g1, gpointer g2)
{
    return (strcmp ((char *) g1, (char *) g2));
}


/*
 * CompareDates
 *
 * Compare two dates and return negative value if first date is
 * less than the second date, positive value if first date is
 * greater than the second date, and zero if the two dates are
 * the same.
 */
gint CompareDates (gpointer g1, gpointer g2)
```

```
{
    typDate *d1 = (typDate *) g1;
    typDate *d2 = (typDate *) g2;

    /* --- Year has highest priority --- */
    if (d1->year == d2->year) {

        /* --- Years are same, what about months? --- */
        if (d1->month == d2->month) {

            /* --- Return difference between days --- */
            return (d1->day - d2->day);
        } else {

            /* --- Months are different - calculate delta --- */
            return (d1->month - d2->month);
        }
    } else {

        /* --- Years are different - calc delta --- */
        return (d1->year -d2->year);
    }
}


/*
 * GetHitsForHour
 *
 * Function to return the the number of hits for a given
 * time of day.
 */
GetHitsForHour (int nHours, long *hits, long *size)
{
    /* --- If the clock is out of range --- */
    if (nHours < 0 || nHours > 23) {

        *hits = 0;
        *size = 0;

    } else {
        *hits = timeOfDayHits[nHours];
        *size = timeOfDaySize[nHours];
```

```
    }
}

/*
 * Init
 *
 * Initialize data so that the log file can be read in.
 * Create trees necessary to store the log data.
 */
int Init ()
{
    int i;

    /* --- Create the trees that will store the information --- */
    fileTree = g_tree_new (CompareStrings);
    dateTree = g_tree_new (CompareDates);
    userTree = g_tree_new (CompareStrings);

    /* --- Clear out the hits by hour --- */
    for (i = 0; i < 24; i++) {
        timeOfDayHits[i] = 0;
        timeOfDaySize[i] = 0;
    }
}



/*
 * FreeURLs
 *
 * Free the memory associated with the URLs.
 * This is a traverse callback
 */
gint FreeURLs (gpointer key, gpointer value, gpointer data)
{
    typStat *info;

    info = (typStat *) value;
    free (info->sURL);
    g_free (info);
    return (0);
}
```

```
/*
 * FreeDates
 *
 * Free information allocated to keep track of the
 * traffic by date.  This is a traverse callback.
 */
gint FreeDates (gpointer key, gpointer value, gpointer data)
{
    typDateInfo *info;

    info = (typDateInfo *) value;
    g_free (info->date);
    g_free (info);
    return (0);
}



/*
 * FreeResources
 *
 * Free the memory used in the tree nodes and then free the
 * trees.
 */
FreeResources ()
{
    /* --- Free data stored in the tree --- */
    g_tree_traverse (userTree, FreeURLs, G_IN_ORDER, NULL);
    g_tree_traverse (dateTree, FreeDates, G_IN_ORDER, NULL);
    g_tree_traverse (fileTree, FreeURLs, G_IN_ORDER, NULL);

    /* --- Free the trees --- */
    g_tree_destroy (userTree);
    g_tree_destroy (dateTree);
    g_tree_destroy (fileTree);

    /* --- Clear pointers --- */
    userTree = NULL;
    dateTree = NULL;
    fileTree = NULL;
}
```

14.4.3 bitmaps.c

列表构件使用图形来显示站点、每天和每小时的通信流量。不是为像元图建立几个 .xpm 文件，而是用程序动态地快速生成 .xpm 文件。

这使得改变像元图的宽度变得十分简单。不必重新画图，就可以改变像元图的宽度。注意，因为需要显示的图是水平的，所以，像元图的影像数据和像元图的每一行数据是一致的。你可以利用这一特点重复利用同一字符串作为像元图每一行。

```
/*
 * CreateBarBitmap
 *
 * Create a pixmap with the characteristics desired.
 *
 * height - height to make the pixmap
 * width - width to make the pixmap
 * size - how big the bar should be
 * sColor - color of the filled-in area
 */
char **CreateBarBitmap(int height, int width, int size, char *sColor)
{
    char **sBitmap;
    char *newbuffer;
    char buffer[88];
    int i;

    /*---Create the room for the data---*/
    sBitmap = g_malloc((height + 1 + 2) * sizeof(gchar *));

    /*---Create pixmap header - height/width/colors/# chars---*/
    sprintf(buffer, "%d %d 2 1", width, height);
    sBitmap[0] = g_strdup(buffer);

    /*---Define the "none" color---*/
    sBitmap[1] = g_strdup("  c None");

    /*---Define the filled-in color---*/
    sprintf(buffer, "X c %s", sColor);
    sBitmap[2] = g_strdup(buffer);

    /*---Fill in the buffer with the size---*/
    strcpy(buffer, "");
```

```
    for (i = 0; i < size; i++) {

        strcat (buffer, "X");
    }

    /* --- Leave the remainder not filled in --- */
    while (i < width) {
        strcat (buffer, " ");
        i++;
    }

    /* --- Pad with blank --- */
    strcat (buffer, " ");

    /* --- Copy the string --- */
    newbuffer = g_strdup (buffer);

    /* --- Make it the data for all the strings --- */
    for (i = 3; i < height + 3; i++) {
        sBitmap[i] = newbuffer;
    }

    /* --- Return the creation --- */
    return (sBitmap);
}


/*
 * FreeBarBitmap
 *
 * Free all the memory that was allocated to create the
 * bitmap.
 */
FreeBarBitmap (char ** bitmap)
{
    g_free (bitmap[0]);
    g_free (bitmap[1]);
    g_free (bitmap[2]);
    g_free (bitmap[3]);
    g_free (bitmap);
}
```

14.4.4　Notebook.c

当将数据装入树后，这些子程序建立列表，并在每个列表构件中显示数据。将构件建立在不同的页上，用户可以通过点击标记切换页。文本被显示，最大流量被计算。这之后，才可以显示图象。因为图象依赖于最大流量，没有最大流量就没法计算，所以最大流量是必须的。

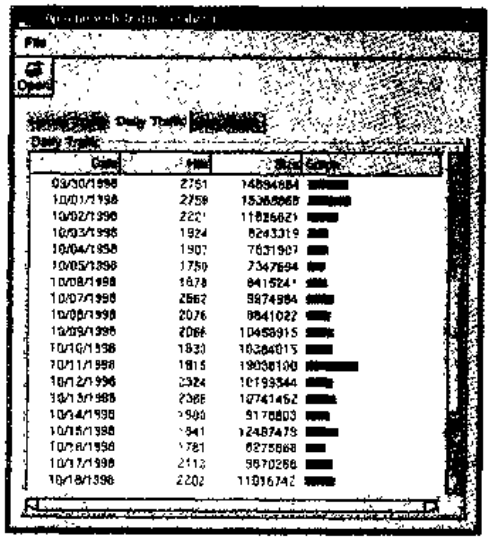


图 14.4　每天的通信流量

```
/*
 * Auth: Eric Harlow
 * File:Notebook.C.
 *
 * Create a sample notebook application
 */

#include <gtk/gtk.h>
#include "logtypes.h"


extern GTree *dateTree;
extern GTree *userTree;

GtkWidget *hourlyPage = NULL;
GtkWidget *dailyPage = NULL;
GtkWidget *userPage = NULL;
GtkWidget *hourlyCList = NULL;
```

```
GtkWidget *dailyCList = NULL;
GtkWidget *userCList = NULL;


typedef struct {
    GtkWidget *widget;
    long nMaxSize;
    long row;
} typGraphInfo;

/*
 * Titles displayed on the clist for the various pages
 */
char *szHourlyTitles[] = {"Hour","Hits","Size","Graph"};
char *szDailyTitles[] = {"Date","Hits","Size","Graph"};
char *szUserTitles[] = {"User","Hits","Size","Graph"};

#define NUM_GRAPHS 21
GdkPixmap *pixmapGraph[NUM_GRAPHS];
GdkBitmap *mask[NUM_GRAPHS];

char **CreateBarBitmap(int height, int width, int size, char *sColor);


/*
 * GeneratePixmaps
 *
 * Generate the pixmaps for all the sizes of horizontal bars
 * that are supported.
 */
GeneratePixmaps(GtkWidget *widget)
{
    int i;
    gchar **pixmap_d;
    GdkColor *transparent = NULL;

    /* --- For each possible graph --- */
    for(i = 0; i < NUM_GRAPHS; i++) {

        /* --- Get the data for the graph --- */
        pixmap_d = CreateBarBitmap(9, 65, i * 3, "#ff0000");
```

```
        /* --- Create a pixmap --- */
        pixmapGraph[i] = gdk_pixmap_create_from_xpm_d (
                        widget->window,
                        &mask[i], NULL,
                        (gpointer) pixmap_d);

        /* --- Free the data --- */
        FreeBarBitmap (pixmap_d);
    }
}


/*
 * AddPage
 *
 * Add a page to the notebook
 *
 * notebook - existing notebook
 * szName - name to give to the new page
 */
GtkWidget *AddPage (GtkWidget *notebook, char *szName)
{
    GtkWidget *label;
    GtkWidget *frame;

    /* --- Create a label from the name --- */
    label = gtk_label_new (szName);
    gtk_widget_show (label);

    /* --- Create a frame for the page --- */
    frame = gtk_frame_new (szName);
    gtk_widget_show (frame);

    /* --- Add a page with the frame and label --- */
    gtk_notebook_append_page (GTK_NOTEBOOK (notebook), frame, label);

    return (frame);
}
/*
 * CreateNotebook
 *
 * Create a new notebook and add pages to it.
```

```
 *
 * window - window to create the notebook in.
 */
void CreateNotebook (GtkWidget * window)
{
    GtkWidget * widget;
    GtkWidget * box1;
    GtkWidget * notebook;

    /* --- Create the notebook --- */
    notebook = gtk_notebook_new ();

    /* --- Listen for the switch page event --- */
    gtk_signal_connect (GTK_OBJECT (notebook), "switch_page",
                        GTK_SIGNAL_FUNC (PageSwitch), NULL);

    /* --- Make sure tabs are on top --- */
    gtk_notebook_set_tab_pos (GTK_NOTEBOOK (notebook), GTK_POS_TOP);

    /* --- Add notebook to vbox --- */
    gtk_box_pack_start (GTK_BOX (window), notebook, TRUE, TRUE, 0);

    /* --- Give notebook a border --- */
    gtk_container_border_width (GTK_CONTAINER (notebook), 10);

    /* --- Add pages to the notebook --- */
    hourlyPage = AddPage (notebook, "Hourly Traffic");
    dailyPage = AddPage (notebook, "Daily Traffic");
    userPage = AddPage (notebook, "User Traffic");

    /* --- Show everything --- */
    gtk_widget_show_all (window);
}


/*
 * PopulatePages
 *
 * Populates the pages on the notebook with the information.
 * Populates the hourly page, daily page, and the user page.
 *
 * Frees the data used to generate the page when done.
```

```
 */
PopulatePages ()
{
    /* --- Free clist data if already used --- */
    if (userCList) {
        gtk_clist_clear (GTK_CLIST (userCList));
    }
    if (hourlyCList) {
        gtk_clist_clear (GTK_CLIST (hourlyCList));
    }
    if (dailyCList) {
        gtk_clist_clear (GTK_CLIST (dailyCList));
    }

    /* --- Populate each of the fields --- */
    PopulateHourly ();
    PopulateDaily ();
    PopulateUser ();

    /* --- Free the resources generated by parselog --- */
    FreeResources ();
}



/*
 * PopulateHourly
 *
 * Populates the clist with the hourly information.
 * Assumes that the trees are fully populated with
 * data ready to be picked.
 */
PopulateHourly ()
{
    gchar *strValue[4];
    int i;
    int ix;
    long hits;
    long size;
    gchar buffer0[88];
    gchar buffer1[88];
    gchar buffer2[88];
```

```
long nMaxSize = 0;

/* --- Here's the array used to insert into clist --- */
strValue[0] = buffer0;
strValue[1] = buffer1;
strValue[2] = buffer2;

/* --- This is NULL because it's a pixmap --- */
strValue[3] = NULL;

/* --- If clist not created yet... --- */
if (hourlyCList == NULL) {

    /* --- Create the clist with four columns --- */
    hourlyCList = gtk_clist_new_with_titles (4, szHourlyTitles);

    /* --- Make sure titles are visible --- */
    gtk_clist_column_titles_show (GTK_CLIST (hourlyCList));

    /* --- Set the column widths --- */
    gtk_clist_set_column_width (GTK_CLIST (hourlyCList), 0, 80);
    gtk_clist_set_column_width (GTK_CLIST (hourlyCList), 1, 80);
    gtk_clist_set_column_width (GTK_CLIST (hourlyCList), 2, 80);
    gtk_clist_set_column_width (GTK_CLIST (hourlyCList), 3, 40);

    /* --- Set the justification on each of the columns --- */
    gtk_clist_set_column_justification (GTK_CLIST (hourlyCList),
                                        0, GTK_JUSTIFY_RIGHT);
    gtk_clist_set_column_justification (GTK_CLIST (hourlyCList),
                                        1, GTK_JUSTIFY_RIGHT);
    gtk_clist_set_column_justification (GTK_CLIST (hourlyCList),
                                        2, GTK_JUSTIFY_RIGHT);

    /* --- Add the clist to the correct page --- */
    gtk_container_add (GTK_CONTAINER (hourlyPage), hourlyCList);
}

/* --- Generate a row for each hour of the day --- */
for (i = 0; i < 24; i++) {
```

```
    /* --- Show the time - like 3:00 --- */
    sprintf (strValue[0], "%d:00", i);

    /* --- Get # of hits for that hour --- */
    GetHitsForHour (i, &hits, &size);

    /* --- Display hit count and byte count --- */
    sprintf (strValue[1], "%ld", hits);
    sprintf (strValue[2], "%ld", size);

    /* --- Add the data to the clist --- */
    gtk_clist_append (GTK_CLIST (hourlyCList), strValue);

    /* --- Keep track of max byte count --- */
    if (size > nMaxSize) {
        nMaxSize = size;
    }
}


/*
 * Now that the clist is generated, we need to go back
 * and add the horizontal graph to the clist.  Couldn't do
 * it earlier because we didn't know the max.
 */

/* --- Every hour of the day --- */
for (i = 0; i < 24; i++) {

    /* --- Get hits for the hour --- */
    GetHitsForHour (i, &hits, &size);

    /* --- Calculate how big graph should be --- */
    ix = (size * NUM_GRAPHS-1) / nMaxSize;

    /* --- Display that graph in the clist --- */
    gtk_clist_set_pixmap (GTK_CLIST (hourlyCList),
          i, 3, (GdkPixmap *) pixmapGraph[ix], mask[ix]);
}

/* --- Show the clist --- */
gtk_widget_show_all (GTK_WIDGET (hourlyCList));
```

```
{

/*
 * ShowDateInfo
 *
 * Show information about the traffic on a particular
 * day.  Dumps the information into the clist thet
 * represents the daily graph.
 *
 * This is called by the tree traverse callback!
 */
gint ShowDateInfo (gpointer key, gpointer value, gpointer data)
{
    char *strValue[4];
    typDateInfo *dateInfo;
    long *pnMax;
    char buffer0[88];
    char buffer1[88];
    char buffer2[88];
    int i;

    /* --- Get info passed it --- */
    dateInfo = (typDateInfo *) value;
    pnMax = (long *) data;

    /* --- Set up structures to populate clist --- */
    strValue[0] = buffer0;
    strValue[1] = buffer1;
    strValue[2] = buffer2;
    strValue[3] = NULL;

    /* --- Fill in the date in the first column --- */
    sprintf (strValue[0], "%02d/%02d/%4d", dateInfo->date->month,
                                           dateInfo->date->day,
                                           dateInfo->date->year);

    /* --- Fill in the hits and byte count --- */
    sprintf (strValue[1], "%ld", dateInfo->nHits);
    sprintf (strValue[2], "%ld", dateInfo->nSize);
```

```
    /* --- Append the data into the clist --- */
    gtk_clist_append (GTK_CLIST (dailyCList), strValue);

    /* --- Keep track of the maximum value --- */
    if (*pnMax < dateInfo->nSize) {

        *pnMax = dateInfo->nSize;
    }

    /* --- 0 => keep on trucking --- */
    return (0);
}



/*
 * ShowUserInfo
 *
 * Shows information about a user (no graphs) but keeps track
 * of the maximum byte count so that the graphs can be generated.
 *
 * This is called as the traverse tree callback.
 */
gint ShowUserInfo (gpointer key, gpointer value, gpointer data)
{
    char *strValue[4];
    typStat *info;
    long *pnMax;
    char buffer0[88];
    char buffer1[88];
    char buffer2[88];
    int i;

    /* --- Get information passed in --- */
    info = (typStat *) value;
    pnMax = (long *) data;

    /* --- Buffers to append data --- */
    strValue[0] = buffer0;
    strValue[1] = buffer1;
    strValue[2] = buffer2;
    strValue[3] = NULL;
```

```
    /* --- Update the URL in first column --- */
    sprintf (strValue[0], "%s", info->sURL);

    /* --- Update bytes and size in next column --- */
    sprintf (strValue[1], "%ld", info->nHits);
    sprintf (strValue[2], "%ld", info->nSize);

    /* --- Add the data to the clist --- */
    gtk_clist_append (GTK_CLIST (userCList), strValue);

    /* --- Keep track of the maximum size --- */
    if (info->nSize > *pnMax) {
        *pnMax = info->nSize;
    }

    return (0);
}


/*
 * DisplayGraph
 *
 * Display the daily graph in the clist.
 *
 * Called as a tree traverse callback.
 */
gint DisplayGraph (gpointer key, gpointer value, gpointer data)
{
    int ix;
    typGraphInfo *graphInfo = (typGraphInfo *) data;
    typDateInfo *dateInfo = (typDateInfo *) value;

    /* --- Figure out which graph to display based on size --- */
    ix = (dateInfo->nSize * NUM_GRAPHS-1) / graphInfo->nMaxSize;

    /* --- Set the pixmap in the clist to this one --- */
    gtk_clist_set_pixmap (GTK_CLIST (graphInfo->widget),
                          graphInfo->row, 3, pixmapGraph[ix], mask[ix]);

    /* --- Next row to display --- */
    graphInfo->row++;
```

```
    /* --- Continue... --- */
    return (0);
}



/*
 * PopulateDaily
 *
 * Populates the clist with the data from the tree.
 * Assumes that the data in the tree has been fully
 * populated.
 */
PopulateDaily ()
{
    gchar * strValue[4];
    int i;
    long hits;
    long size;
    long nMaxDaily;
    gchar buffer0[88];
    gchar buffer1[88];
    gchar buffer2[88];
    int nRow = 0;
    typGraphInfo graphInfo;

    /* --- Create tha table --- */
    strValue[0] = buffer0;
    strValue[1] = buffer1;
    strValue[2] = buffer2;

    /* --- NULL - graphic is going here --- */
    strValue[3] = NULL;

    /* --- If the clist has not been created yet... --- */
    if (dailyCList == NULL) {

        /* --- Create the clist --- */
        dailyCList = gtk_clist_new_with_titles (4, szDailyTitles);

        /* --- Make sure titles are being shown --- */
```

```
        gtk_clist_column_titles_show (GTK_CLIST (dailyCList));

        /* --- Set the column widths --- */
        gtk_clist_set_column_width (GTK_CLIST (dailyCList), 0, 80);
        gtk_clist_set_column_width (GTK_CLIST (dailyCList), 1, 80);
        gtk_clist_set_column_width (GTK_CLIST (dailyCList), 2, 80);

        /* --- Set the column justifications --- */
        gtk_clist_set_column_justification (GTK_CLIST (dailyCList),
                                            0, GTK_JUSTIFY_RIGHT);
        gtk_clist_set_column_justification (GTK_CLIST (dailyCList),
                                            1, GTK_JUSTIFY_RIGHT);
        gtk_clist_set_column_justification (GTK_CLIST (dailyCList),
                                            2, GTK_JUSTIFY_RIGHT);

        /* --- Add the clist to the notebook page --- */
        gtk_container_add (GTK_CONTAINER (dailyPage), dailyCList);
    }

    /* --- set max to zero --- */
    nMaxDaily = 0;

    /*
     * --- Traverse tree and display the textual information
     *     while gathering the maximum so thet the graph can
     *     be displayed
     */
    g_tree_traverse (dateTree, ShowDateInfo, G_IN_ORDER, &nMaxDaily);

    /* --- Information for displaying of the graph --- */
    graphInfo.nMaxSize = nMaxDaily;
    graphInfo.widget = dailyCList;
    graphInfo.row = 0;

    /* --- Re-traverse the tree and display graphs --- */
    g_tree_traverse (dateTree, DisplayGraph, G_IN_ORDER, &graphInfo);

    /* --- Show the clist now --- */
    gtk_widget_show_all (GTK_WIDGET (dailyCList));
}
```

```
/*
 * DisplayUserGraph
 *
 * Display the graph for each user.
 * This is called from a traverse tree - it's a callback with the
 * data passed into it.
 *
 * value - contains information about this user's activity
 * data - contains information about the graph, incl. widget and max
 */
gint DisplayUserGraph (gpointer key, gpointer value, gpointer data)
{
    int ix;
    typGraphInfo *graphInfo = (typGraphInfo *) data;
    typStat *statInfo = (typStat *) value;

    /* --- How big should the graph be? --- */
    ix = (long) (((double) statInfo->nSize * NUM_GRAPHS-1) /
                            graphInfo->nMaxSize);

    /* --- Set the pixmap to an appropriate size --- */
    gtk_clist_set_pixmap (GTK_CLIST (graphInfo->widget),
                          graphInfo->row, 3, pixmapGraph[ix], mask[ix]);

    /* --- Go to the next row --- */
    graphInfo->row++;

    return (0);
}


/*
 * PopulateUser
 *
 * Populate the user graph with the information about each user's
 * Web site traffic.  The display is created in two parts.  The
 * first part displays the text data and computes the necessary
 * values for the second part to display the graph.
 */
PopulateUser ()
{
    gchar *strValue[4];
```

```
int i;
long hits;
long size;
gchar buffer0[88];
gchar buffer1[88];
gchar buffer2[88];
int nRow = 0;
long nMax;
typGraphInfo graphInfo;

/* --- Buffered values --- */
strValue[0] = buffer0;
strValue[1] = buffer1;
strValue[2] = buffer2;
strValue[3] = NULL;

/* --- If there's no user clist yet --- */
if (userCList == NULL) {

    /* --- Create the clist with titles --- */
    userCList = gtk_clist_new_with_titles (4, szUserTitles);

    /* --- Show titles --- */
    gtk_clist_column_titles_show (GTK_CLIST (userCList));

    /* --- Show width of columns --- */
    gtk_clist_set_column_width (GTK_CLIST (userCList), 0, 80);
    gtk_clist_set_column_width (GTK_CLIST (userCList), 1, 80);
    gtk_clist_set_column_width (GTK_CLIST (userCList), 2, 80);

    /* --- Justify columns --- */
    gtk_clist_set_column_justification (GTK_CLIST (userCList),
                                        0, GTK_JUSTIFY_LEFT);
    gtk_clist_set_column_justification (GTK_CLIST (userCList),
                                        1, GTK_JUSTIFY_RIGHT);
    gtk_clist_set_column_justification (GTK_CLIST (userCList),
                                        2, GTK_JUSTIFY_RIGHT);

    /* --- Add clist to page --- */
    gtk_container_add (GTK_CONTAINER (userPage), userCList);
}
```

```
    /* --- Traverse the tree to show text info and get max --- */
    nMax = 0;
    g_tree_traverse (userTree, ShowUserInfo, G_IN_ORDER, &nMax);

    /* --- Populate structure for graphical tree traversal --- */
    graphInfo.nMaxSize = nMax;
    graphInfo.widget = userCList;
    graphInfo.row = 0;

    /* --- Display graphs --- */
    g_tree_traverse (userTree, DisplayUserGraph, G_IN_ORDER, &graphInfo);

    gtk_widget_show_all (GTK_WIDGET (userCList));
}
```

14.5　小结

现在，你可以使用更复杂的构件建立更有趣的应用程序。GtkTree 构件显示的信息可用树来很好加以描述：分支是可以显示的，或者将它折叠起来只显示你想看见的部分，GtkNotebook 构件一般用于显示多页程序，也可以隐藏标记，使它变成由程序控制。GtkCList 一般用于显示来自数据库的数据，正文和图像共存的列表看起来会更赏心悦目。

第 15 章　创建自己的构件

在开发应用程序的过程中有时会发现，现有的构件不能完全满足我们的需要。例如：也许游戏需要雷达屏幕构件，或者应用程序需要显示多种图形。即使 GTK + 包含了大量的构件，它也不可能满足所有情况的需要。幸运的是，GTK + 提供了一个界面使得开发者可以扩充原有的构件集合。

15.1　理解构件

学习如何创建新的构件的最好途径是阅读可供使用的 GTK + 源代码。探索如何创建新事物乃是最好的学习方法。这一章将通过例子说明 GTK + 是如何创建的。

GTK + 具有面向对象的特征。但它是用 C 语言编写的，而 C 却不是面向对象的语言。所以，当我们想要创建面向对象的构件时需要更多的知识。要理解所有创建构件的技巧确实有一定困难。可是，一旦构件被创建，以面向对象的方式使用它却是非常容易的。创建构件需要两个文件。一个是建立和定义构件的实际源代码文件，另一个是定义数提结构和原型的头文件。

15.1.1　继承

可以在已有的构件的基础上创建自己的构件，也可以从头做起。如果一个已有的构件已经实现许多必要的特征，那么在它的基础上进行编程会更加容易。例如：触发按钮（GtkToggleButton）以按钮构件（GtkButton）作为基础。触发按钮继承了按钮构件的许多特征。并以它自己的特征补充了按钮构件的特征。

15.1.2　从头做起

创建构件也可以从头做起，但这种方法需要做更多的工作。许多从头设计的构件以 Gtkwidget 作为基础构件。Gtkwidget 提供了最少的必需的基础以便开发者添加新的功能。另外一些从头设计的构件则继承 GtkMisc 的功能。和 GtkWidget 相比，基于 GtkMisc 的构件消耗资源比较少，因为它们不直接和 X Window 发生联系。它们使用父构件的 X Window 来完成绘制工作。由于它们不直接和 X Window 发生联系，因此它们也有许多限制。例如：标号（GtkLabel）不能接受鼠标事件。为使它接受鼠标事件，只好将它放在 GtkEventBox 内。标号是从 GtkMisc 派生而来，它是只显示的构件，通常不处理鼠标事件。

15.2　构件如何工作

只要有可能，设计新构件应该利用其他构件作为基础。这样做可以借鉴其他开发人员的

程序,也可以减少代码的不必要的膨胀。这一部分将介绍一些已经存在的构件,并分析它们的代码以理解构件的不同组成部分。这里的许多例子是关于按钮的,但它在许多方面和其他构件是类似的。

15.2.1　包含文件

构件的包含文件定义构件所必需的数据结构和构件函数的原型。需要记住的是,一个包含文件不能被包含多次。为了避免这样的问题,构件的头文件使用特殊的标志来标记它们自己。此外,包含文件可能与 C++一起使用,所以它们被标记为 C 子程序供连接程序使用。Gtk_button.h 是这样开始的:

```
#ifndef __GTK_BUTTON_H__
#define __GTK_BUTTON_H__
#include <gdk/gdk.h>
#include <gtk/gtkcontainer.h>

#ifdef __cplusplus
extern "C" {
#pragma }
#endif /* __cplusplus */

/*
 *
 * include file stuff goes here.
 *
 */
```

包含文件应该用下面的代码作为结束以与文件的开始部分相呼应。

```
#ifdef __cplusplus
}
#endif /* __cplusplus */


#endif /* __GTK_BUTTON_H__ */
```

当然,如果你在建立构件,应该用自己的标号代替这个定义。

15.2.2　宏

头文件还定义用于新构件的宏。我们经常使用像 GTK_BUTTON 这样的宏来把 GtkWidget 转换成 GtkButton。它和其他宏在下面定义,按钮构件定义了三个宏:

```
#define GTK_BUTTON(obj)  (GTK_CHECK_CAST ((obj), GTK_TYPE_BUTTON, GtkButton))
#define GTK_BUTTON_CLASS(klass)     (GTK_CHECK_CLASS_CAST ((klass), GTK_TYPE_BUTTON,
```

```
GtkButtonClass))
#define GTK_IS_BUTTON(obj)    (GTK_CHECK_TYPE((obj), GTK_TYPE_BUTTON))
```

在这种情况下,GTK_TYPE_BUTTON 定义为:

```
#define GTK_TYPE_BUTTON    (gtk_button_get_type())
```

使用 GtkButton 的程序都使用 GTK_BUTTON,而其他两个宏主要由 Gtk_Button 构件程序内部使用。

15.2.3 数据结构

下一步确定需要什么样的数据结构。对这一步,你需要确定响应什么信号,以及在结构中保存什么数据。GtkButton 需要一个小的结构。

```
struct _GtkButton
{
  GtkContainer container;

  GtkWidget *child;

  guint in_button : 1;
  guint button_down : 1;
};
```

GtkButtonx 结构定义用于每个按钮的局部数据。在这个结构中的第一项中必须定义这一构件是从哪一个父构件派生而来。在这里是 GtkContainer。其余的数据是用于按钮本身的数据。

还需要建立按钮的类。就像按钮构件有局部数据和信息一样,类需要保存对特定类型所有构件共同需要的信息,如可用信号等。按钮的类如下:

```
struct _GtkButtonClass
{
  GtkContainerClass parent_class;

  void (* pressed)  (GtkButton *button);
  void (* released) (GtkButton *button);
  void (* clicked)  (GtkButton *button);
  void (* enter)    (GtkButton *button);
  void (* leave)    (GtkButton *button);
};
```

值得注意的是,在结构中先定义父类,后而跟以函数指针的清单。在新构件的类结构中必须先定义父类结构。将这些指针看作是 C 中的虚函数。要到构件创建之后才会被定义。但是如果另一个构件使用 GtkButton 作为基类(如 GtkToggleButton),它可以改变按钮的行为。

你也需要使用 typedef 来定义这些结构(不带下划线)。

```
Typedef struct _GtkButton       GtkButton;
Typedef struct _GtkButtonClass GtkButtonClass;
```

15.2.4 原型

头文件的最后部分定义了由应用程序使用的任何原型。至少原型应该定义 new 构件函数和那些处理构件的函数。GtkButton 构件具有下面的原型定义:

```
GtkType     gtk_button_get_type       (void);
GtkWidget * gtk_button_new            (void);
GtkWidget * gtk_button_new_with_label (const gchar * label);
void        gtk_button_pressed        (GtkButton * button);
void        gtk_button_released       (GtkButton * button);
void        gtk_button_clicked        (GtkButton * button);
void        gtk_button_enter          (GtkButton * button);
void        gtk_button_leave          (GtkButton * button);
```

15.2.5 C代码

C 文件比头文件复杂。幸运的是,许多代码可以从已有构件的代码中照搬,或者稍加修改。第一步是需要列举出那些由构件定义的信号(不用包含那些已经由基础构件定义的信号)。GtkButton 构件定义了下列信号,最后以 LAST_SIGNAL 指示符作为结束。

```
Enum {
  PRESSED
  RELEASED
  CLICKED
  ENTER
  LEAVE
  LAST_SIGNAL
};
```

然而,以 GtkButton 作为基类的 GtkToggleButton 只需要定义专门用于触发按钮的信号。触发按钮需要增加"toggled"信号。因此,它定义新信号如下:

```
enum {
  TOGGLED
  LAST_SIGNAL
};
```

每个构件都需要一个 get_type 函数,它给 GTK+ 提供有关这个构件的信息。将这些信息组合在 GtkTypeInfo 结构中,然后传递给 Gtk_type_unique 来惟一地识别这个新构件。

GtkTypeInfo 需要下列的信息:

构件的名称

- 对象的大小(例如,GtkButton 构件有多大?)
- 类的大小(例如,GtkButtonClass 有多大?)
- 类的初始化函数
- 对象的初始化函数
- 设置变元的函数
- 取得变元的函数

对于按钮,这一结构定义如下:

```
GtkTypeInfo button_info =
      {
        "GtkButton",
        sizeof (GtkButton),
        sizeof (GtkButtonClass),
        (GtkClassInitFunc) gtk_button_class_init,
        (GtkObjectInitFunc) gtk_button_init,
        (GtkArgSetFunc) gtk_button_set_arg,
        (GtkArgGetFunc) NULL,
      };
```

将这个结构和父类的类型一起传递给 Gtk_type_unique 函数来产生构件的惟一的标识符(ID)。构件标识符只被产生一次,并且将返回的值加以保存。当为按钮调用 get_type 函数时会用到它。下面是 get_type 函数:

```
GtkType
gtk_button_get_type (void)
{
  static GtkType button_type = 0;

    /* --- If ID not generated yet --- */
    if (!button_type)
      {
        GtkTypeInfo button_info =
        {
          "GtkButton",
          sizeof (GtkButton),
          sizeof (GtkButtonClass),
          (GtkClassInitFunc) gtk_button_class_init,
          (GtkObjectInitFunc) gtk_button_init,
          (GtkArgSetFunc) gtk_button_set_arg,
          (GtkArgGetFunc) NULL,
```

```
        };

        button_type = gtk_type_unique (gtk_container_get_type (),
                                       &button_info);
    }

    return button_type;
}
```

下一步定义 gtk_button_class_init 和 gtk_button_init 函数。

15.2.6　类的初始化

在 get_type 函数内部定义的 class_init 函数用来建立构件的类结构。构件的类结构定义适用于这种类型的构件的共同信息。信息包含新的信号和重新定义(需要覆盖的)旧信号。增加新的按钮信号使用了静态数组,它的定义如下:

```
static guint button_signals[LAST_SIGNAL] = {0};
```

将 LAST_SIGNAL 定义为信号列举的一部分。这个数组将用构件的信号识别符所填充。这些信号识志符由 gtk_signal_new 函数产生。

按钮的类初始化被分为几个部分。对重要的程序段综合说明如下。第 1 步是从构件的类结构中获得关于父类的信息。

```
static void gtk_button_class_init (GtkButtonClass *klass)
{
  GtkObjectClass *object_class;
  GtkWidgetClass *widget_class;
  GtkContainerClass *container_class;

  object_class = (GtkObjectClass*) klass;
  widget_class = (GtkWidgetClass*) klass;
  container_class = (GtkContainerClass*) klass;

  parent_class = gtk_type_class (gtk_container_get_type ());
```

第 2 步是创建新的信号。button_signals 数组已经在前面建立。现在必须用信号填充它。由 gtk_signal_new 函数创建每一个信号,并提供数组内的位置索引值。定义 gtk_signal_new 函数如下:

```
gint gtk_signal_new (const gchar *name,
                     GtkSignalRunType run_type,
                     GtkType object_type,
                     gint function_offset,
```

```
                    GtkSignalMarshaller marshaller,
                    GtkType return_val,
                    gint nparams,
                    [parameter types]);
```

"pressed"和"clicked"信号只是在名称和信号位移值上稍有不同。它们都使用了缺省的marshaller,并具有零参数。

```
button_signals[PRESSED] =
    gtk_signal_new ("pressed",
                    GTK_RUN_FIRST,
                    object_class->type,
                    GTK_SIGNAL_OFFSET (GtkButtonClass, pressed),
                    gtk_signal_default_marshaller,
                    GTK_TYPE_NONE, 0);

button_signals[CLICKED] =
    gtk_signal_new ("clicked",
                    GTK_RUN_FIRST,
                    object_class->type,
                    GTK_SIGNAL_OFFSET (GtkButtonClass, clicked),
                    gtk_signal_default_marshaller,
                    GTK_TYPE_NONE, 0);
```

然后,将信号加到对象类:

```
Gtk_object_class_add_signal (object_class, button_signals,LAST_SIGNAL);
```

构件也可以选择覆盖任何父类中的任何信号。在这里,button 选择覆盖许多构件信号和部分容器信号。

```
widget_class->activate_signal = button_signals[CLICKED];
widget_class->map = gtk_button_map;
widget_class->unmap = gtk_button_unmap;
widget_class->realize = gtk_button_realize;
widget_class->draw = gtk_button_draw;
widget_class->draw_focus = gtk_button_draw_focus;
widget_class->draw_default = gtk_button_draw_default;
widget_class->size_request = gtk_button_size_request;
widget_class->size_allocate = gtk_button_size_allocate;
widget_class->expose_event = gtk_button_expose;
widget_class->button_press_event = gtk_button_button_press;
widget_class->button_release_event = gtk_button_button_release;
widget_class->enter_notify_event = gtk_button_enter_notify;
```

```
widget_class->leave_notify_event = gtk_button_leave_notify;
widget_class->focus_in_event = gtk_button_focus_in;
widget_class->focus_out_event = gtk_button_focus_out;
container_class->add = gtk_button_add;
container_class->remove = gtk_button_remove;
container_class->foreach = gtk_button_foreach;
```

最后，填充按钮信号。“clicked”信号被设置成空，因为构件不关心这样的信号。虽然，开发人员在对按钮编程时可能会关心这一信号。

```
klass->pressed = gtk_real_button_pressed;
klass->released = gtk_real_button_released;
klass->clicked = NULL;
klass->enter = gtk_real_button_enter;
klass->leave = gtk_real_button_leave;
}
```

15.2.7　发送信号

可以通过 gtk_signal_emit 或 gtk_signal_emit_by_name 函数发送信号。通常，在构件内部使用 gtk_signal_emit，因为它访问信号表，并获得信号的标识符。因为 gtk_signal_emit_by_name 函数使用信号名称而不是识别符，所以它可以在构件外部使用。在按钮的代码中，可以调用 gtk_button_clicked 函数发送“clicked”信号。它所作的只是调用 gtk_signal_emit 函数。

```
Void gtk_button_clicked (GtkButton *button)
{
    gtk_signal_emit (GTK_OBJECT (button), button_signals[CLICKED]);
}
```

如果你想要事件发生，使回呼函数处理这一事件，你可以调用 gtk_signal_emit_by_name 函数。

```
gtk_signal_emit_by_name (GTK_OBJECT (button),"changed");
```

当发送信号时，它将会被传播给所有的信号处理程序。可以在一个信号处理程序中用 gtk_signal_emit_stop_by_name 函数来阻止信号的传播。当你想过滤一些信号时，你就可以这样做。例如：你想过滤掉一些键。你可以创建“key_press_event”回呼函数。当按下这些键时，它调用 gtk_signal_emit_stop_by_name 函数阻止其他信号处理程序得到“key_press_event”信号。

15.2.8　初始化函数

init 函数初始化按钮以供使用。这一步包含对按钮的数据结构中的数据进行初始化，有

时创建另一些构件。GtkButton 的初始化函数是简单的。因为它所做的全部工作只是初始化。

```
static void gtk_button_init (GtkButton * button)
{
  GTK_WIDGET_SET_FLAGS (button, GTK_CAN_FOCUS);

  button->child = NULL;
  button->in_button = FALSE;
  button->button_down = FALSE;
}
```

另外一些构件初始化工作会复杂一些。GtkFileSelection 就比较复杂,因为它包含对所有对话框的子构件的建立。在文件选择对话框中需要建立列表框构件和按钮构件。使得建立文件选择对话框的工作很复杂。如果你创建一个简单的构件,你的初始化函数大致和按钮构件的初始化函数差不多。

15.3 创建构件

在写好数据结构、类的初始化和构件初始化的代码之后,最重要的一步就是写允许开发人员建立构件的程序。对 GtkButton 讲,它有两个函数:gtk_button_new 和 gtk_button_new_with_label。gtk_button_new 函数创建构件包括:取得类型和建立这一类型的实例。下面是建立构件的标准代码的一部分:

```
GtkWidget *
gtk_button_new (void)
{
  return GTK_WIDGET (gtk_type_new (gtk_button_get_type ()));
}
```

更有趣的程序是建立带标号的按钮的程序。创建一个按钮和一个标号,并把标号放在按钮上。这并不需要关于 GtkButton 内部的知识。它的代码如下:

```
GtkWidget *
gtk_button_new_with_label (const gchar * label)
{
  GtkWidget * button;
  GtkWidget * label_widget;

  button = gtk_button_new ();
  label_widget = gtk_label_new (label);
  gtk_misc_set_alignment (GTK_MISC (label_widget), 0.5, 0.5);
```

```
    gtk_container_add (GTK_CONTAINER (button), label_widget);
    gtk_widget_show (label_widget);

    return button;
}
```

15.3.1　创建图形构件

从头开始创建构件不很容易，但也不是特别难。拥有 GTK + 的源代码的积极意义在于：当你建立新构件时，可以从已有的大量构件的建立过程中借鉴许多有用的知识。为了了解如何从头开始建立构件，我们建立一个显示简单图形的简单图形构件。当考虑从头建立构件时，应该了解它与别的构件的类似之处。在现在的情况下，可以用绘图区构件作为模型，因为它提供完成绘图所需要的最少函数。你可以扩充从学习绘图区构件得到的知识来创建图形构件。

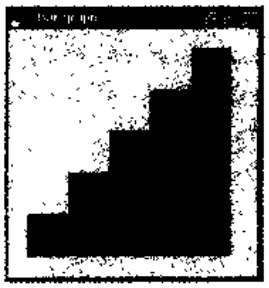


图 15.1　图形构件

15.3.2　头文件

这个小的头文件为图形构件简单描叙了使构件工作所必要的部分。构件在它的数据结构的称为 values 的数组内保存图形数据以及图形数据元素的数目。

```
/*
 * File: gtkgraph.h
 * Auth: Eric Harlow
 */
#ifndef __GTK_GRAPH_H__
#define __GTK_GRAPH_H__


#include <gdk/gdk.h>
#include <gtk/gtkvbox.h>


#ifdef __cplusplus
extern "C" {
```

```
# endif /* __cplusplus */


/*
 * --- Macros for conversion and type checking
 */
# define GTK_GRAPH(obj) \
  GTK_CHECK_CAST (obj, gtk_graph_get_type (), GtkGraph)
# define GTK_GRAPH_CLASS(klass) \
  GTK_CHECK_CLASS_CAST (klass, gtk_graph_get_type, GtkGraphClass)
# define GTK_IS_GRAPH(obj) \
  GTK_CHECK_TYPE (obj, gtk_graph_get_type ())

/*
 * --- Defining data structures
 */

typedef struct _GtkGraph        GtkGraph;
typedef struct _GtkGraphClass   GtkGraphClass;

/*
 * Here's the graph data
 */
struct _GtkGraph
{
    GtkWidget vbox;

    gint *values;
    gint num_values;
};


/*
 * Here's the class data
 */
struct _GtkGraphClass
{
  GtkWidgetClass parent_class;
};


/*
```

```
 * Function prototypes
 */
GtkWidget * gtk_graph_new (void);
void gtk_graph_size (GtkGraph *graph, int size);
void gtk_graph_set_value (GtkGraph *graph, int index, int value);


#ifdef __cplusplus
}
#endif /* __cplusplus */


#endif /* __GTK_GRAPH_H__ */
```

15.3.3　图形程序

图形程序定义从 GtkWidget 继承的图形构件。图形构件需要覆盖原有构件用于 draw、expose、realize 和 size_request 的代码，以改变原有构件的行为。GtkGraph 也覆盖删除对象信号，在调用父类删除回呼函数以前释放已经分配的内存。两个填充图象的函数是 Gtk_graph_size 和 Gtk_graph_set_value。主要的函数是 draw 函数，它由 gtk_graph_draw 函数调用。代码如下：

```
/*
 * File: GtkGraph.c
 * Auth: Eric Harlow
 *
 * Simple gtk graphing widget
 */

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <gtk/gtk.h>
#include "gtkgraph.h"

static GtkWidgetClass *parent_class = NULL;

/*
 * forward declarations:
 */
static void gtk_graph_class_init (GtkGraphClass *class);
static void gtk_graph_init (GtkGraph *graph);
```

```
static void gtk_graph_realize (GtkWidget *widget);
static void gtk_graph_draw (GtkWidget *widget, GdkRectangle *area);
static void gtk_graph_size_request (GtkWidget *widget,
                                    GtkRequisition *req);
static gint gtk_graph_expose (GtkWidget *widget, GdkEventExpose *event);
static void gtk_graph_destroy (GtkObject *object);


/*
 * gtk_graph_get_type
 *
 * Internal class. Used to define the GtkGraph class to GTK+
 */
guint gtk_graph_get_type (void)
{
  static guint graph_type = 0;

    /* --- If not created yet --- */
    if (!graph_type) {

        /* --- Create a graph_info object --- */
        GtkTypeInfo graph_info =
            {
              "GtkGraph",
              sizeof (GtkGraph),
              sizeof (GtkGraphClass),
              (GtkClassInitFunc) gtk_graph_class_init,
              (GtkObjectInitFunc) gtk_graph_init,
              (GtkArgSetFunc) NULL,
              (GtkArgGetFunc) NULL,
            };

        /* --- Tell GTK+ about it - get a unique identifying key --- */
        graph_type = gtk_type_unique (gtk_widget_get_type (), &graph_info);
    }
    return graph_type;
}


/*
 * gtk_graph_class_init
 *
```

```
 * Override any methods for the graph class that are needed for
 * the graph class to behave properly.  Here, the functions that
 * cause painting to occur are overridden.
 *
 * class - object definition class.
 */
static void gtk_graph_class_init (GtkGraphClass *class)
{
    GtkObjectClass *object_class;
    GtkWidgetClass *widget_class;

    /* --- Get tha widget class --- */
    object_class = (GtkObjectClass *) class;
    widget_class = (GtkWidgetClass *) class;
    parent_class = gtk_type_class (gtk_widget_get_type ());

    /* --- Override object destroy --- */
    object_class->destroy = gtk_graph_destroy;

    /* --- Override these widget methods --- */
    widget_class->realize = gtk_graph_realize;
    widget_class->draw = gtk_graph_draw;
    widget_class->size_request = gtk_graph_size_request;
    widget_class->expose_event = gtk_graph_expose;
}

/*
 * gtk_graph_init
 *
 * Called each time a graph item gets created.
 * This initializes fielde in our structure.
 */
static void gtk_graph_init (GtkGraph *graph)
{
    GtkWidget *widget;

    widget = (GtkWidget *) graph;

    /* --- Initial values --- */
    graph->values = NULL;
    graph->num_values = 0;
}
```

```
/*
 * gtk_graph_new
 *
 * Create a new GtkGraph item
 */
GtkWidget * gtk_graph_new (void)
{
  return gtk_type_new (gtk_graph_get_type ());
}


/*
 * gtk_graph_realize
 *
 * Associate the widget with an X Window.
 *
 */
static void gtk_graph_realize (GtkWidget *widget)
{
  GtkGraph *darea;
  GdkWindowAttr attributes;
  gint attributes_mask;

  /* --- Check for failures --- */
  g_return_if_fail (widget != NULL);
  g_return_if_fail (GTK_IS_GRAPH (widget));

  darea = GTK_GRAPH (widget);
  GTK_WIDGET_SET_FLAGS (widget, GTK_REALIZED);

  /* --- attributes to create the window --- */
  attributes.window_type = GDK_WINDOW_CHILD;
  attributes.x = widget->allocation.x;
  attributes.y = widget->allocation.y;
  attributes.width = widget->allocation.width;
  attributes.height = widget->allocation.height;
  attributes.wclass = GDK_INPUT_OUTPUT;
  attributes.visual = gtk_widget_get_visual (widget);
  attributes.colormap = gtk_widget_get_colormap (widget);
  attributes.event_mask = gtk_widget_get_events (widget) | GDK_EXPOSURE_MASK;
```

```
  /* --- We're passing in x, y, visual and colormap values --- */
  attributes_mask = GDK_WA_X | GDK_WA_Y | GDK_WA_VISUAL | GDK_WA_COLORMAP;

  /* --- Create the window --- */
  widget->window = gdk_window_new (gtk_widget_get_parent_window (widget),
                                   &attributes, attributes_mask);
  gdk_window_set_user_data (widget->window, darea);

  widget->style = gtk_style_attach (widget->style, widget->window);
  gtk_style_set_background (widget->style, widget->window, GTK_STATE_NORMAL);
}

/*
 * gtk_graph_size
 *
 * Custom method to set the size of the graph.
 */
void gtk_graph_size (GtkGraph *graph, int size)
{
    g_return_if_fail (graph != NULL);
    g_return_if_fail (GTK_IS_GRAPH (graph));

    graph->num_values = size;
    graph->values = g_realloc (graph->values, sizeof (gint) * size);
}


/*
 * gtk_graph_set_value
 *
 * Custom method to set the size.
 */
void gtk_graph_set_value (GtkGraph *graph, int index, int value)
{
    g_return_if_fail (graph != NULL);
    g_return_if_fail (GTK_IS_GRAPH (graph));
    g_return_if_fail (index < graph->num_values && index >= 0);

    graph->values[index] = value;
}
```

```
/*
 * gtk_graph_draw
 *
 * Draw the widget.  Draws the widget based on the
 * number of values in the bar graph.
 */
static void gtk_graph_draw (GtkWidget *widget, GdkRectangle *area)
{
    GtkGraph *graph;
    int width;
    int height;
    int column_width;
    int max = 0;
    int i;
    int bar_height;

    /* --- Check for obvious problems --- */
    g_return_if_fail (widget != NULL);
    g_return_if_fail (GTK_IS_GRAPH (widget));

    /* --- Make sure it's a drawable widget --- */
    if (GTK_WIDGET_DRAWABLE (widget)) {

        graph = GTK_GRAPH (widget);
        if (graph->num_values == 0) {
            return;
        }

        /* --- Get height and width --- */
        width = widget->allocation.width - 1;
        height = widget->allocation.height - 1;

        /* --- Calculate width of the columns --- */
        column_width = width / graph->num_values;

        /* --- Find the max value --- */
        for (i = 0; i < graph->num_values; i++) {
            if (max < graph->values[i]) {
                max = graph->values[i];
            }
        }
```

```
        /* --- Display each bar graph --- */
        for (i = 0; i < graph->num_values; i++) {

            bar_height = (graph->values[i] * height) / max;

            gdk_draw_rectangle (widget->window,
                                widget->style->fg_gc[GTK_STATE_NORMAL],
                                TRUE,
                                (i * column_width),
                                height-bar_height,
                                (column_width-2),
                                bar_height);
        }
    }
}


/*
 * gtk_graph_size_request
 *
 * How big should the widget be?
 * It can be modified.
 */
static void gtk_graph_size_request (GtkWidget *widget,
                                    GtkRequisition *req)
{

    req->width = 200;
    req->height = 200;
}


/*
 * gtk_graph_expose
 *
 * The graph widget has been exposed and needs to be painted.
 *
 */
static gint gtk_graph_expose (GtkWidget *widget, GdkEventExpose *event)
{
    GtkGraph *graph;

    /* --- Do error checking --- */
```

```
    g_return_val_if_fail (widget != NULL, FALSE);
    g_return_val_if_fail (GTK_IS_GRAPH (widget), FALSE);
    g_return_val_if_fail (event != NULL, FALSE);

    if (event->count > 0) {
        return (FALSE);
    }

    /* --- Get the graph widget --- */
    graph = GTK_GRAPH (widget);

    /* --- Clear the window --- */
    gdk_window_clear_area (widget->window, 0, 0,
                           widget->allocation.width,
                           widget->allocation.height);

    /* --- Draw the graph --- */
    gtk_graph_draw (widget, NULL);
}

/*
 * gtk_graph_destroy
 *
 * Destroy the widget and free up any allocated memory.
 * When done, call the parent destroy to make sure any
 * memory allocated by the parent is freed.
 */
static void gtk_graph_destroy (GtkObject * object)


    GtkGraph * graph;

    /* --- Check type --- */
    g_return_if_fail (object != NULL);
    g_return_if_fail (GTK_IS_GRAPH (object));

    /* --- Convert to graph object --- */
    graph = GTK_GRAPH (object);

    /* --- Free memory --- */
    g_free (graph->values);
```

```
    /* --- Call parent destroy --- */
    GTK_OBJECT_CLASS (parent_class)->destroy (object);
}
```

15.3.4　使用构件

现在你可以使用图形构件在窗口内显示简单的图形了。下面的例子建立图形构件，并用待显示的图形填充它。

```
/*
 * File: main.c
 * Auth: Eric Harlow
 *
 * Example showing custom widget
 */

#include <gtk/gtk.h>
#include "gtkgraph.h"

/*
 * CloseAppWindow
 *
 * The window is closing down.  Need to shut down GTK+.
 */
gint CloseAppWindow (GtkWidget *widget, gpointer *data)
{
    gtk_main_quit ();

    return (FALSE);
}



/*
 * main - program begins here
 */
int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *graph;

    /* --- GTK+ initialization --- */
```

```
gtk_init (&argc, &argv);

/* --- Create the top-level window --- */
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

gtk_window_set_title (GTK_WINDOW (window), "Bar graph");

/* --- You should always remember to connect the delete_event
       to the main window. --- */
gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                    GTK_SIGNAL_FUNC (CloseAppWindow), NULL);

/* --- Give the window a border --- */
gtk_container_border_width (GTK_CONTAINER (window), 20);

/*
 * --- Create a graph
 */

/* --- Create a new graph --- */
graph = gtk_graph_new ();

/* --- Show the graph --- */
gtk_widget_show (graph);

/* --- Set the number of elements in the graph --- */
gtk_graph_size (GTK_GRAPH (graph), 5);

/* --- Set the height of each of the graph elements --- */
gtk_graph_set_value (GTK_GRAPH (graph), 0, 5);
gtk_graph_set_value (GTK_GRAPH (graph), 1, 10);
gtk_graph_set_value (GTK_GRAPH (graph), 2, 15);
gtk_graph_set_value (GTK_GRAPH (graph), 3, 20);
gtk_graph_set_value (GTK_GRAPH (graph), 4, 25);
gtk_widget_draw (graph, NULL);

/*
 * --- Make the main window visible
 */
gtk_container_add (GTK_CONTAINER (window), graph);
gtk_widget_show (window);
```

```
    gtk_main();
    exit(0);
}
```

15.4　小结

创建构件是简单的。构件为应用程序提供了比其内部实现要简单得多的界面。构件可以以继承其他构件的方式建立，也可以从头开始建立。从头开始建立构件需要做更多的工作，但也带来了更大的灵活性。

实用技术丛书

丛书特点：

专家专为专业人士而写

经过专家组审阅测评

具有极高的知识含金量

内容精炼，领域集中

丛书内容：

Windows NT、BackOffice

Linux、Unix、Solaris

网络技术及协议

读者对象：

该技术领域的新手

更关心解决方案、故障排除、系统维护等实用技术的人员

希望能快速有效地解决技术问题，而不关心理论及细节的人士

读者定位：

中级、高级、专业级

丛书包括：

书代号	书名	出版时间
TP55920	实用技术：规化 Windows 2000	2000.1
TP54270	实用技术：WINDOWS NT TCP/IP 实用指南	99.10
TP54840	实用技术：精通 WINDOWS NT DNS	99.6
TP55890	实用技术：Windows NT 网络管理——如何降低总成本	99.11
TP56880	实用技术：Windows NT 注册管理参考手册	99.12
TP56890	实用技术：SQL Server 7.0 系统管理	99.12
TP56420	实用技术：Exchange 5.5 系统管理	99.10
TP56020	实用技术：Exchange 服务器的实现(汉英对照)	99.10
TP56800	实用技术：开发 Linux 应用——用 GTK+和 GDK 开发 Linux 图形用户界面应用	2000.1
TP54590	Windows NT——性能监视、测试和调整	99.8

如果您有兴趣编写或翻译以上内容图书，请与以下编辑联系：

吴　源、周宏敏　　Email：wuyuan@proxy.phei.com.cn

电　话：6815.9577 转 465、463　　传　真：6815.9025

通　信：北京 173 信箱电子工业出版社(100036)

买书请与邮购部联系：

钱　宇　　电话：6827.8599、6827.9077　　传真：6815.9021

汇款及通信：北京 173 信箱电子工业出版社邮购部(100036)